TUGAS AKHIR – KI091391

RANCANG BANGUN APLIKASI PENDETEKSI *URL* BERBAHAYA PADA *STREAM* TWITTER

IDHAM MARDI PUTRA
NRP 5110 100 024

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., PhD.

Dosen Pembimbing II
Baskoro Adi Pratomo, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA 2014

UNDERGRADUATE THESIS – KI091391

DEVELOPMENT AND DESIGN OF APPLICATION FOR DETECTING SUSPICIOUS URLs IN TWITTER STREAM

IDHAM MARDI PUTRA
NRP 5110 100 024

First Supervisor
Royyana Muslim Ijtihadie, S.Kom., M.Kom., PhD.

Second Supervisor
Baskoro Adi Pratomo, S.Kom., M.Kom.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA 2014

LEMBAR PENGESAHAN

RANCANG BANGUN APLIKASI PENDETEKSI *URL* BERBAHAYA PADA *STREAM* TWITTER

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Berbasis Jaringan
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :
IDHAM MARDI PUTRA
NRP . 5110 100 024

Disetujui oleh Dosen Pembimbing Tugas Akhir:

1. Royyana Muslim Ijtihadie, S.Kom., M.Kom., PhD. ..... (Pembimbing 1)
   NIP: 197708242006041001

2. Baskoro Adi Pratomo, S.Kom., M.Kom. ..... (Pembimbing 2)
   NIP: 510000003

SURABAYA
Juli, 2014

RANCANG BANGUN APLIKASI PENDETEKSI *URL* BERBAHAYA PADA *STREAM* TWITTER

Nama : **Idham Mardi Putra**
NRP : **5110 100 024**
Jurusan : **Teknik Informatika – FTIf ITS**
Dosen Pembimbing I : **Royyana Muslim Ijtihadie, S.Kom., M.Kom., PhD.**
Dosen Pembimbing II : **Baskoro Adi Pratomo, S.Kom., M.Kom.**

Abstrak

Perkembangan teknologi internet yang sangat pesat merupakan hal yang tidak dapat dipungkiri. Setiap orang dapat bertukar informasi di manapun dan kapanpun tanpa terbatas ruang dan waktu. Salah satu contoh dari kemajuan teknologi di era globalisasi ini tentang bagaimana cara pengguna bertukar pesan dan informasi yakni menggunakan media sosial Twitter.

Twitter merupakan salah satu layanan berbagi informasi atau bertukar pesan yang kurang dari 140 karakter, yang biasa disebut dengan tweets. Informasi yang disebarkan melalui tweet biasanya tidak hanya sekedar informasi berupa kata-kata, melainkan pengguna dapat menyebarkan suatu URL yang twitter sendiri mempunyai layanan penyusutan URL, sehingga dengan memanfaatkan Twitter sebagai salah satu media sosial yang popular di dunia, beberapa orang menyelewengkan layanan tersebut sebagai aksi kriminalitas dengan melakukan spam dan mendistribusikan URL yang mencurigakan.

Untuk mencegah tindakan kriminalitas tersebut, maka diperlukan suatu aplikasi pendeteksi URL berbahaya pada tweet bagi pengguna Twitter. Perlindungan yang dilakukan dengan memberikan peringatan jika terdapat URL yang termasuk dalam kategori berbahaya. Sistem peringatan ini ditanamkan ke dalam peramban yang digunakan pengguna untuk mencegah pengguna mengunjungi URL yang telah disebarkan melalui tweet ketika

sedang mengunjungi situs Twitter. Aplikasi ini mendeteksi URL yang mempunyai pengalihan secara berantai, selain itu dikarenakan penyerang mempunyai keterbatasan sumber daya penyebaran URL pada satu akun, sehingga mereka seringkali membuat banyak akun dan dapat menyebarkan URL yang sama, maka aplikasi ini dapat mengecek kemiripan antara akun palsu yg dibuat penyerang tersebut dengan mengecek konteks informasi pada akun tersebut.

Dari hasil uji coba terhadap sistem deteksi ini, kecepatan deteksi pada sekali pengecekkan rata-rata membutuhkan total waktu selama 531.79 detik. Tingkat akurasi dari hasil uji coba mencapai diatas 90% dengan menggunakan metode klasifikasi decision tree.

Kata kunci: Decision Tree, Keamanan Informasi, Twitter, URL, URL Redirectional.

DEVELOPMENT AND DESIGN OF APPLICATION FOR DETECTING SUSPICOUS URLs IN TWITTER STREAM

Name	**: Idham Mardi Putra**
NRP	**: 5110 100 024**
Department	**: Teknik Informatika – FTIf ITS**
Supervisor I	**: Royyana Muslim Ijtihadie, S.Kom., M.Kom., PhD.**
Supervisor II	**: Baskoro Adi Pratomo, S.Kom., M.Kom.**

Abstract

The rapid development of internet technology is inevitable. Everyone can exchange information anywhere and anytime without limited by space and time. One of the examples of technological advancement in current globalization era is that how we, as users, exchanging messages and information namely through the use of Twitter.

Twitter provides service, either sharing of information or exchanging a message which is less than equal 140 characters, generally we call it tweets. Information spread through Twitter are generally not only information in the form of plain texts, but also users are able to spread the URLs which is Twitter itself has a service of URL shrinking. So that, by taking this advantage had by Twitter, one of well-known social media in the world, some irresponsible people are going to abuse this service to do criminal activity by spamming and by distributing suspicious URL.

To prevent this criminal activity, therefore it is needed to invent an application for users which is able to detect malicious URL in the tweets. The protection is done by giving an alert if there is a URL which is acknowledged as potentially malicious. This warning system is planted in the device used by user to prevent user from accessing malicious URLs which had been spread through tweets while user is connecting to Twitter's site. This

application detects URLs which have strange chain redirection, moreover due to attacker's limited resource of spreading URL in one account, they are sometimes going to make more accounts and spreading the same URL, thus this application is able to check the resemblance among these pre-made fake accounts by checking the information context in these accounts.

By the reference of experiment results toward this detection system, the detection speed at once checking needs total time of 513.79 seconds in average. The accuracy level of this experiment reaches above 90% by using decision tree classification method.

Keywords: Decision Tree, Information Security, Twitter, URL, URL Redirection.

KATA PENGANTAR

بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

Alhamdulillahirabbil'alamin, segala puji bagi Allah SubhanahuWata'alla, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul "**Rancang Bangun Aplikasi Pendeteksi *URL* Berbahaya Pada *Stream* Twitter**" dengan baik.

Pengerjaan Tugas Akhir ini merupakan suatu kesempatan yang sangat berharga bagi penulis, karena dengan pengerjaan Tugas Akhir ini, penulis bisa memperdalam, meningkatkan, serta mengimplementasikan apa yang telah didapatkan penulis selama menempuh perkuliahan di Teknik Informatika ITS.

Dalam pelaksanaan dan pembuatan Tugas Akhir ini, tentunya penulis ingin mengucapkan terima kasih, memberikan penghormatan, dan apresasi setinggi-tingginya kepada pihak-pihak yang telah berperan penting bagi penulis dalam menyelesaikan Tugas Akhir ini, maupun dalam proses menempuh pendidikan pada jenjang Strata Satu Jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember Surabaya. Melalui lembar ini, penulis ingin secara khusus menyampaikan ucapan terimakasih yang sebesar-besarnya kepada:

1. Allah SWT atas limpahan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir ini dengan baik.
2. Kedua orang tua penulis yang telah mencurahkan, perhatian, doa, semangat, serta dukungan dalam penyelesaian tugas akhir ini.
3. Mbak Amy, kakak yang selalu memberikan motivasi dan inspirasi sehingga dapat menyelesaikan Tugas Akhir ini dengan baik.
4. Bapak Royyana Muslim Ijtihadie dan Bapak Baskoro Adi Pratomo yang telah bersedia meluangkan waktu untuk memberikan saran, masukan, kritik, serta bimbingan dalam peyelesain tugas akhir ini.

5. Ibu Nanik Suciati, S.Kom., M.Kom., Dr.Eng. selaku ketua jurusan Teknik Informatika ITS dan selaku dosen wali penulis, serta segenap Bapak/Ibu dosen Teknik Informatika yang telah memberikan ilmunya kepada penulis.
6. Novita Nata Wardhanie, kekasih penulis yang selalu memberikan *support* dan motivasi untuk fokus pada pengerjaan tugas akhir ini.
7. Mas Aldo Aditya Alase, Happy Ayu Christianty, Raditya Andre Nurwityanto, Puspa Arty Ghaisany, Dewa Ayu Sri Mertiani, Dimas Prabowo, Nandez Darras, Ramadhan Satya Putra, Abdurrazak Baihaqi, dan segenap rekan - rekan perjuangan lainnya atas saran, kerjasama dan masukannya mengenai proses penyelesaian tugas akhir dan penulisan buku tugas akhir ini.
8. Seluruh keluarga laboratorium AJK, mas aldo, mas kevin, mas satrio, mas om, mas erico, thiar, pur, surya, eva, happy, rangga, bowo, beha, rama, uyunk, romen, Jordy, tuharum, sam, agus, nandez, vivi, mas aldo, dan pak bas.
9. Seluruh teman Teknik Informatika ITS 2010 dan C-1A.
10. Juga tidak lupa kepada semua pihak yang belum sempat disebutkan satu per satu yang telah membantu terselesaikannya Tugas Akhir ini.

Permintaan maaf terbesar dari penulis atas segala kekurangan yang masih terdapat dari diri penulis, baik yang terdapat pada buku ini maupun hingga selesainya buku ini. Saran dan kritik yang membangun penulis terima dengan tangan terbuka.

Surabaya, Juni 2014

Idham Mardi Putra

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR PERSAMAAN

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

BAB I
PENDAHULUAN

Pada bab ini akan dijelaskan mengenai beberapa hal dasar dalam Tugas Akhir ini yang meliputi latar belakang, rumusan masalah, tujuan dan manfaat pembuatan Tugas Akhir, serta metodologi dan sistematika penulisan buku Tugas Akhir ini.

1.1. Latar Belakang

Twitter merupakan salah satu media sosial dan layanan berbagi informasi yang memberikan fasilitas bagi pengguna untuk saling bertukar pesan kurang dari 140 karakter dengan memanfaatkan jaringan internet, yang biasa disebut dengan *tweets*. Pengguna dapat mendistribusikan *tweet*-nya otomatis ke semua pengguna lain yang telah terdaftar sebagai teman dari pengguna tersebut, yang biasa disebut dengan *followers*. Selain mendistribusikan *tweet*-nya ke semua *followers*, pengguna tersebut juga dapat mengirim sebuah *tweet* ke salah satu pengguna dengan menyebutkan nama pengguna lain yang didahului tanda "@" pada depan nama pengguna tersebut seperti @bob. Pada Twitter, hal ini biasa disebut dengan *mention*. Pengguna melakukan *mention* tidak harus kepada teman yang terdaftar sebagai *followers*, namun juga dapat melakukannya kepada pengguna lain yang bukan terdaftar sebagai *followers.*

Ketika para pengguna Twitter ingin berbagi sebuah *URL* yang diselipkan pada *tweet*-nya, mereka sering kali memanfaatkan layanan untuk menyusutkan *URL* dengan mengurangi panjang dari karakter sebuah *URL*, seperti *bit.ly* dan *tinyurl.com*. hal ini dikarenakan keterbatasan dari jumlah karakter pada tweet tersebut. Sedangkan Twitter sendiri juga mempunyai layanan untuk penyusutan *URL* secara otomatis. Artinya terdapat suatu rantai *URL* yang saling mengalihkan hingga ke suatu domain situs tertentu, dalam hal ini *URL* tersebut bisa dikategorikan *URL* yang dicurigai berbahaya.

Dengan memanfaatkan Twitter sebagai salah satu media sosial yang populer di seluruh penjuru dunia, para penyerang dunia maya (*attacker*) mengambil peluang dengan mendistribusikan *URL* berbahaya tersebut kepada semua pengguna Twitter. Perlu diketahui bahwa, *spam* yang merupakan salah satu bentuk paling umum dari serangan web, menjadi bermunculan di Twitter. *URL* yang telah disisipkan oleh *attacker* pada *tweet* merupakan suatu rantai dari *URL* lain yang mengarah pada halaman tertentu [1].

Berdasarkan pendeteksi *URL* berbahaya pada media sosial seperti Twitter ini sudah pernah diusulkan dengan menggunakan *social honeypot* yang memanfaatkan suatu agen pada *honeyclient* yang biasa disebut dengan *crawler* untuk menangani kasus semacam itu. Namun para *attacker* tidak tinggal diam, mereka menggunakan beberapa teknik untuk menghindari para *crawler* dan meneruskan *URL* tersebut ke *URL* yang umum, seperti *google.com*. Tidak hanya itu, seorang *attacker* dapat dengan mudah membuat akun–akun palsu pada Twitter untuk mendistribusikan *tweet*-nya.

Oleh karena itu, pada tugas akhir ini, dirancang dan diimplementasikan suatu aplikasi sistem pendeteksi *URL* berbahaya pada *stream* Twitter yang diambil dari Twitter *stream* API dengan mengamati rantai *URL* dari suatu *tweet*. Data diambil dan dan dikumpulkan hingga *n-tweets*. Kemudian setelah data dikumpulkan, dicari *entrypoint URL* dengan mencocokkan *redirect URL* satu dengan *URL* yang lain, kemudian dicek dengan menggunakan dua *user-agent* yakni dengan menggunakan *browser normal* dan juga *crawler*. Kemudian dilakukan tahapan ekstraksi fitur dengan cara mengelompokan *URL* dari dua URL atau lebih pada *tweet* lain. Selanjutnya pengekstrakan didasari oleh dua fitur, yakni fitur pertama yang berasal dari korelasi antar rantai *URL* mulai dari panjang rantai *URL*, posisi dari *entry point URL*, banyaknya jumlah inisial *URL* yang berbeda, dan jumlah *landing URL* yang berbeda dari suatu *URL*, dan fitur yang kedua berdasarkan konteks informasi dari *tweet* itu sendiri seperti, jumlah akun Twitter yang berbeda, standar deviasi [2] dari *followers*-

following pengirim, dan kesamaan isi teks *tweet* satu dengan yang lain pada pengirim.

1.2. Rumusan Permasalahan

Adapun yang menjadi rumusan masalah dalam Tugas Akhir ini adalah sebagai berikut:

1. Bagaimana cara menentukan *URL redirect* terindikasi berbahaya atau bukan?
2. Bagaimana membedakan akun pengguna Twitter dengan akun palsu yang dibuat oleh penyerang?
3. Bagaimana menerapkan metode klasifikasi dengan menggunakan *Decision Tree*?
4. Bagaimana melakukan uji coba pada metode yang telah diimplementasikan?
5. Bagaimana mengambil kesimpulan akhir untuk menentukan apakah *URL* yang telah didistribusikan melalui *tweet* berbahaya atau tidak?
6. Bagaimana cara mengirimkan suatu notifikasi apabila ditemukan suatu *URL* yang berbahaya?

1.3. Batasan Masalah

Dari permasalahan yang telah diuraikan di atas, terdapat beberapa batasan masalah pada Tugas Akhir ini, yaitu:

a) Aplikasi ini menggunakan *crawler* yang statis, sehingga hanya mampu menangani HTPP *redirection*.
b) Untuk melakukan pengambilan *tweet* yang mengandung *URL*, aplikasi ini membutuhkan *history* dari *tweet* sebelumnya sehingga aplikasi ini tidak bisa berjalan secara *real time*, tetapi harus mengambil dan mengumpulkan data tweet yang mengandung *URL* terlebih dahulu.
c) Hanya 10 fitur yang digunakan sebagai *feature classification*, yakni: panjang rantai dari suatu *URL*, frekuensi dari *entrypoint URL*, letak dari *entrypoint URL*, jumlah inisial *URL* pada *entrypoint* yang sama, jumlah pengirim *URL* dengan *Entrypoint* yang sama, simpangan baku pada *followers* pengirim,

simpangan baku pada *friends* pengirim, simpangan baku pada rasio *followers-friends* pengirim, dan kemiripan teks *tweet* pada pengirim.

1.4. Tujuan dan Manfaat

Tujuan dari pembuatan tugas akhir ini adalah membuat aplikasi untuk mendeteksi adanya *URL* yang berbahaya pada *tweet*, selain itu mencegah terjadinya pendistribusian *URL* pada *tweet* yang berkepanjangan pada akun–akun palsu yang dibuat penyerang dan melakukan dengan cara *spam*.

Manfaat pembuatan tugas akhir ini adalah untuk menghindarkan pengguna dari kejahatan pada dunia maya yang menyerang sosial media khususnya Twitter yang mengandung unsur *URL* yang dicurigai berbahaya.

1.5. Metodologi

Adapun langkah-langkah yang ditempuh dalam pengerjaan Tugas Akhir ini adalah sebagai berikut:

1. Penyusunan proposal Tugas Akhir

Tahap awal untuk memulai pengerjaan TA adalah penyusunan proposal Tugas Akhir. Pada proposal tersebut dijelaskan secara garis besar tentang bagaimana cara mengindentifikasi apakah suatu *URL* yang terkandung pada *tweet* merupakan berbahaya.

2. Studi literatur

Pada tahapan ini, penulis melakukan pengumpulan informasi yang diperlukan untuk pengerjakan tugas akhir. Literatur diperoleh oleh dari berbagai dokumen baik buku referensi, *paper*, maupun *internet*.

3. Perancangan sistem

Tahap ini merupakan perancangan sistem dengan menggunakan studi literatur dan mempelajari konsep aplikasi yang akan dibuat. Dengan berbekal teori, metode dan informasi yang sudah terkumpul pada tahap sebelumnya diharapkan dapat membantu dalam proses perancangan sistem.

4. Implementasi perangkat lunak

Tahap ini merupakan implementasi rancangan sistem yang telah dibuat. Tahap ini merealisasikan apa yang terdapat pada tahapan perancangan yang telah dibuat sebelumnya sehingga menjadi sebuah rancang bangun sistem yang sesuai dengan apa yang telah direncanakan.

5. Pengujian dan evaluasi

Aplikasi akan diuji setelah selesai diimplemetasikan menggunakan skenario yang sudah dipersiapkan. Pengujian dan evaluasi akan dilakukan dengan melihat kesesuaian dengan perencanaan. Dengan melakukan pengujian dan evaluasi dimaksudkan juga untuk mengevaluasi jalannya program, mencari masalah yang mungkin timbul dan mengadakan perbaikan jika terdapat kesalahan.

6. Penyusunan buku Tugas Akhir.

Pada tahap ini disusun laporan Tugas Akhir sebagai dokumentasi pelaksanaan Tugas Akhir, yang mencakup seluruh konsep, teori, implementasi, serta hasil yang telah dikerjakan.

1.6. Sistematika Penulisan

Buku tugas akhir ini disusun dengan sistematika penulisan sebagai berikut:

BAB I. PENDAHULUAN

Bab ini berisi latar belakang, permasalahan, tujuan, batasan permasalahan, metodologi, dan sistematika penulisan.

BAB II. TINJAUAN PUSTAKA

Bab ini berisi penjelasan secara detail mengenai dasar-dasar teori penunjang yang digunakan untuk mendukung penyelesaian Tugas Akhir.

BAB III. PERANCANGAN PERANGKAT LUNAK

Bab ini berisi tentang perancangan sistem, diagram alir, dan perancangan antarmuka yang akan dibuat.

BAB IV. IMPLEMENTASI PERANGKAT LUNAK

Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa *pseudocode* dan *screenshot* aplikasi.

BAB V. EVALUASI DAN UJI COBA

Bab ini menjelaskan kemampuan perangkat lunak dengan melakukan pengujian fungsionalitas dan pengujian performa dalam beberapa skenario.

BAB VI. PENUTUP

Bab ini merupakan bab terakhir yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan dan saran untuk pengembangan perangkat lunak ke depannya.

BAB II
TINJAUAN PUSTAKA

Pada bab ini dijelaskan beberapa hal mengenai teori yang berkaitan dengan sistem yang diimplementasikan. Hal ini ditujukan untuk memberikan gambaran secara umum terhadap sistem yang akan dibuat. Selain itu, hal tersebut berguna untuk menunjang pembuatan sistem sehingga kebutuhannya dapat diketahui.

2.1. Twitter

Twitter adalah layanan jejaring sosial yang memungkinkan penggunanya untuk mengirim dan membaca pesan berbasis teks hingga 140 karakter, yang dikenal dengan sebutan kicauan (*tweet*). Twitter didirikan pada bulan Maret 2006 oleh Jack Dorsey, dan situs jejaring sosialnya diluncurkan pada bulan Juli. Sejak diluncurkan, Twitter telah menjadi salah satu dari sepuluh situs yang paling sering dikunjungi di internet. Di Twitter, pengguna tidak terdaftar hanya dapat membaca *tweet*, sedangkan pengguna terdaftar bisa mengirim *tweet* melalui antarmuka situs web, pesan singkat (SMS), atau melalui berbagai aplikasi untuk perangkat seluler [3].

Melalui Twitter seorang pemilik akun dapat melakukan *update* status atau berinteraksi dengan pemilik akun lainnya. Terdapat beberapa istilah yang umum digunakan pada jejaring sosial Twitter, istilah-istilah tersebut diantaranya:

a. *http://t.co/* merupakan layanan dari Twitter untuk menyusutkan *URL* dan berbentuk unik antara pengguna satu dengan yang lain, hal ini dikarenakan panjang karakter maksimal pada *tweets* hanya 140 karakter, sehingga ketika pengguna mengirimkan *URL* pada *tweet*-nya, Twitter menghasilkan *shorthing URL* tersebut.

b. *Direct message*, lebih dikenal dengan sebutan DM atau pesan yang merupakan sebuah fitur untuk berkirim pesan dari pengirim kepada penerima secara *private*.
c. *Follow*, mengikuti kebiasan seorang pengguna dalam mengirimkan *tweet* atau *update* status.
d. *Hashtag*, ditandai dengan simbol '#' yang digunakan untuk menandai suatu *keyword* atau topik.
e. *Home*, sekumpulan *tweet* dari pemilk akun lain yang di-*follow* oleh pengguna dan ditampilkan secara *real-time*.
f. *Mention*, ditandai dengan simbol '@' yang berarti memberikan pesan atau seruan yang ditujukan kepada pengguna Twitter.
g. *OAuth*, suatu metode untuk memberikan akses kepada pihak ke-3 (dapat berupa sebuah aplikasi) dari seorang pengguna tanpa memberikan sandi yang dimiliki pengguna.
h. *Profile*, halaman Twitter yang berisi semua informasi pengguna dan menampilkan informasi tentang pengguna, serta semua *tweet* yang telah dikirim dari akun pengguna.
i. *Reply*, sebuah *tweet* yang dikirim untuk membalas pesan yang dikirimkan oleh pengguna lainnya dan dimulai dengan *username* '@'.
j. *Timeline*, sekumpulan *tweet* yang ditampilkan secara *real-time*.

Username, digunakan untuk mengidentifikasi seorang pengguna atau pemilik akun Twitter dan setiap *username* bersifat unik dan terdiri dari 15 karakter.

2.2. *URL (Uniform Resource Locator)*

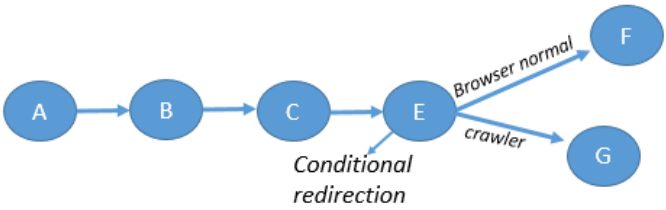
Uniform Resource Locator yang dahulu disebut *Universal Resource Locator* adalah alamat unik sebuah *file* yang dapat diakses di Internet. Secara umum *URL* digunakan untuk mendapatkan *website* yang dikunjungi adalah dengan memasukan *file URL* di baris alamat pada *web browser* [4].

URL pertama kali diciptakan oleh Tim Berners-Lee pada tahun 1991 agar para penulis dokumen mereferensikan pranala ke *World Wide Web*. Sejak tahun 1994, konsep *URL* diperkembangkan dan diistilahkan *Uniform Resource Identifier* (*URI*) agar lebih umum sifatnya. Meskipun begitu, istilah *URL* masih tetap dipergunakan secara luas.

2.3. Analisis *URL Redirect Conditional Chains*

*URL redirec*tion merupakan suatu *URL* yang terlihat bukanlah *URL* yang sebenarnya ingin dituju. *URL* tersebut melakukan banyak pengalihan halaman yang telah ditentukan oleh seorang *attackers*. Biasanya mereka memanfaatkan layanan untuk menyusutkan *URL* dengan mengurangi panjang dari karakter sebuah *URL*, seperti *bit.ly* dan *tinyURL.com*. selain itu pada Twitter itu sendiri juga mempunyai layanan memperkecil ukuran karakter dari suatu *URL* seperti t.co, dan mereka mempunyai halaman yang dapat mengalihkan lebih dari satu tujuan agar membedakan pengunjung yang menggunakan *browser* yang normal atau *Crawler*.

Hal tersebut dipengaruhi oleh *user-agent* yang dipakai pengguna. Sehingga attacker membuat redirect path dari *URL* tersebut apabila *requester* dari *URL* tersebut merupakan browser normal, maka akan diarahkan ke situs yang diinginkan, namun apabila *requester* dari *URL* tersebut merupakan *crawler* maka akan diarahkan ke situs yang jinak seperti *www.google.com* [1]. Hal ini ditunjukan pada Gambar 2.1.



Gambar 2.1 alur *redirect URL* secara kondisional

2.4. Basis Data

Basis data pada *server* digunakan untuk menampung seluruh kebutuhan data pada sistem. Dalam sistem ini akan digunakan basis data MySQL. MySQL merupakan sebuah perangkat lunak *Database Management System* (DBMS) yang sudah banyak digunakan dari sebuah sistem dengan skala kecil sampai skala besar.

2.5. TweetSharp

TweetSharp merupakan library dari Twitter API yang digunakan menggunakan bahasa C#. *Application Programming Interface* atau yang biasa disebut dengan API sendiri merupkan sekumpulan perintah, fungsi, dan protokol yang dapat digunakan oleh programmer saat membangun perangkat lunak untuk sistem operasi tertentu. API memungkinkan *programmer* untuk menggunakan fungsi standar untuk berinteraksi dengan sistem operasi [5].

```
1   using TweetSharp;
2   var service = new TwitterService(_consumerKey,
3   _consumerSecret);
4       service.AuthenticateWith(_accessToken,
5   _accessTokenSecret);
6       var tweets =
7   service.ListTweetsOnHomeTimeline(new
8   ListTweetsOnHomeTimelineOptions());
9   foreach (var tweet in tweets)
10  {
11     Console.WriteLine("{0} says '{1}'",
12  tweet.User.ScreenName, tweet.Text);
13  }
14
```

Gambar 2.2 Contoh kode program menggunakan TweetSharp

Dengan adanya TweetSharp, kita dengan mudah melakukan pengambilan data Twitter pada aplikasi *desktop*, *web* bahkan aplikasi berbasis *mobile*. Kita dapat membangun mulai dari *widget* yang sederhana hingga kompleksitas yang kita inginkan.

Saat ini untuk *library* yang telah digunakan adalah *TweetSharp-Unofficial* versi 2.3.1.2. dimana telah dikembangkan oleh Timothy-Makarov [5].Contoh program sederhana dengan menggunakan TweetSharp Library ditunjukan pada Gambar 2.2.

2.6. User-Agent

User-agent merupakan perangkat lunak yang bertindak sebagai identitas saat melakukan *request* lokasi situs. *User-agent* berupa sederatan kode string informasi yang dikirim ke web server dari browser yang kita gunakan saat mengakses sebuah halaman web [6]. Dalam beberapa kasus kerap kali dalam mengakses sebuah halaman web juga kadang tidak terbuka atau di-*redirect* ke halaman lain, itu dikarenakan *user-agent browser* yang kita gunakan tidak diizinkan oleh *web server* untuk mengakses halaman tersebut. *Web server* juga menggunakan informasi dari *user-agent* pada *browser* dalam mengakses sebuah halaman yang kemudian menyesuaikannya dengan *browser* yang digunakan, misalnya *user-agent* pada peramban Mozilla Firefox akan memunculkan kode string yang ditunjukan pada Gambar 2.3, sedangkan bila menggunakan *crawler* [7] akan memunculkan Gambar 2.4 saat mengakses halaman.

```
    Mozilla/5.0 (Windows NT 6.2; WOW64; rv:29.0)
Gecko/20100101 Firefox/29.0
```

Gambar 2.3 Contoh kode *string user-agent* menggunakan peramban Mozilla Firefox

```
    Mozilla/5.0     (compatible;     Googlebot/2.1;
+http://www.google.com/bot.html)
```

Gambar 2.4 Contoh kode *string user-agent* menggunakan Google *Crawler*

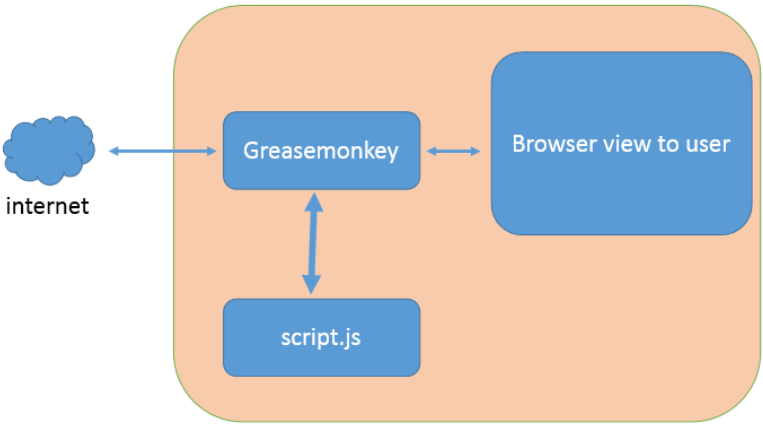
2.7. Greasemonkey Mozilla *add-ons*

Greasemonkey merupakan salah satu *add-ons* peramban Mozila Firefox yang digunakan untuk memanipulasi atau mengustomisasi tampilan web aslinya menjadi web yang dinginkan melalui skrip. Proyek ini dimulai sejak 28 November

2004 dan ditulis oleh Aaron Boodman. Greasemonkey mengijinkan pengguna untuk menginstall skrip sehingga dapat mengubah konten dari halaman *web* yang menggunakan antarmuka *Document Object Model* menjadi seperti yang kita inginkan. Skrip tersebut menggunakan bahasa pemrograman *javascript* [8].

Greasemonkey dapat dibuat maupun diinstall langsung untuk keperluan tertentu, bahkan banyak pihak telah menyalahgunakannya untuk membuat *malware* yang dapat menyimpan data pribadi.

Gambaran umum proses yang dilakukan oleh Greasemonkey ialah ditunjukan pada Gambar 2.5.



Gambar 2.5 Sistem Kerja Greasemonkey *Add-ons*

2.8. JavaScript

JavaScript merupakan bahasa pemrograman yang umum digunakan dalam pemrograman web. Bahasa pemrograman ini didukung pada berbagai peramban, salah satunya Mozilla Firefox. Karena sudah umum digunakan, banyak peramban modern berbasis *desktop* maupun *mobile* saat ini memanamkan dukungan *JavaScript* [9].

JavaScript merupakan bahasa pemrograman yang berjalan pada sisi klien (*client-side scripting*). Hal ini menyebabkan setiap eksekusi perintah dilakukan oleh peramban dimana pengguna mengakses situs. Penggunaan *JavaScript* sendiri berdampingan dengan HTML dan CSS, dimana *JavaScript* dapat digunakan untuk memanipulasi konten dan desain dari situs.

2.9. Algoritma *Decision Tree*

Algoritma *decision tree* atau pohon keputusan, merupakan metode klasifikasi dengan melakukan model prediksi dengan menggunakan struktur pohon atau struktur berhirarki untuk menentukan sebuah keputusan [10]. Konsep dari *decision tree* adalah kemampuan untuk mem-*break down* proses pengambilan keputusan yang kompleks menjadi lebih simpel sehingga pengambilan keputusan akan lebih menginterpretasikan solusi dari permasalahan.

Keleibihan dari metode *decision tree* adalah:

1. Daerah pengambilan keputusan yang sbelumnya kompleks dan sangat global, dapat diubah menjadi lebih simpel dan spesifik.
2. Eliminasi perhitungan-perhitungan yang tidak diperlukan, karena ketika menggunakan metode decision tree maka sampel yang diuji hanya berdasarkan kriteria atau kelas tertentu.
3. Fleksibel untuk memilih fitur dari internal nodes yang berbeda, fitur yang terpilih akan membedakan suatu kriteria dibandingkan kriteria yang lain dalam *node* yang sama. Kefleksibelan metode decision tree ini meningkatkan kualitas keputusan yang dihasilkan jika dibandingkan ketika menggunakan metode penghitungan satu tahap yang lebih konvensional.

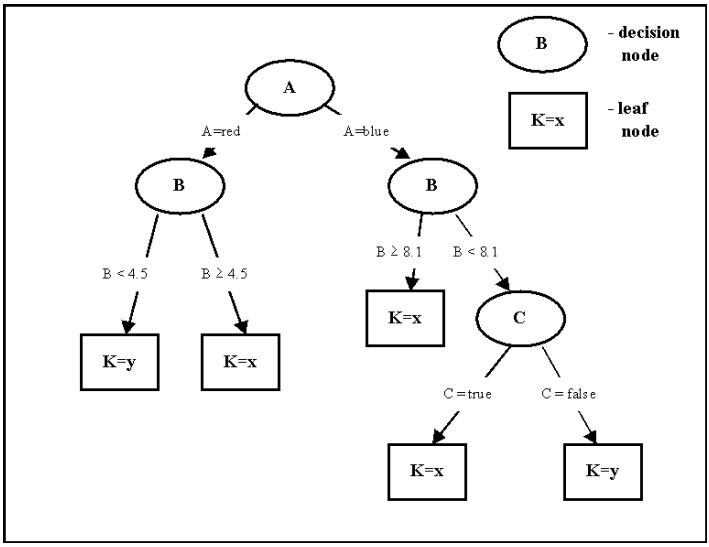
Kekurangan dari metode *decision tree*, yakni:

1. Terjadi *overlap* terutama ketika kelas-kelas dan kriteria yang digunakan jumlahnya sangat banyak. Hal tersebut juga dapat menyebabkan meningkatnya waktu

pengambilan keputusan dan jumlah *memory* yang diperlukan.

2. Pengakumulasian jumlah *error* dari setiap level dalam sebuah pohon keputusan yang besar.
3. Kesulitan dalam mendesain *decision tree* yang optimal. Hasil kualitas keputusan yang didapatkan dari metode *decision tree* sangat tergantung pada bagaimana pohon tersebut didesain.

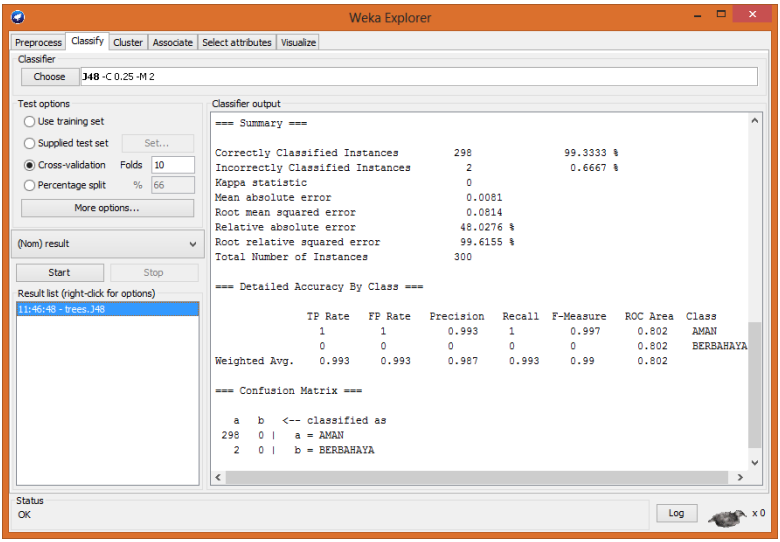
Untuk lebih jelasnya dari pengimplementasian metode klasifikasi *decision tree*, dapat ditunjukan pada Gambar 2.6.



Gambar 2.6 Metode klasifikasi *decision tree*

2.10. Weka

Weka merupakan kakas yang populer dalam bidang *data mining*. Kakas ini dikembangkan oleh University of Waikato, New Zealand. Di dalam weka terdapat fitur untuk *data pre-processing*, klasifikasi, regresi, *clustering*, aturan asosiasi, serta visualisasi data [11].



Gambar 2.7 Perhitungan akurasi Weka

Weka dikembangkan menggunakan bahasa pemrograman Java. Aplikasi tersedia dalam bentuk GUI (*Graphical User Interface*) serta bisa digunakan sebagai pustaka pemrograman. Melihat dari kebutuhan sistem yang mengimplementasikan bahasa pemrograman C# dalam pengembangan, Weka digunakan untuk mendapatkan hasil akurasi yang cukup stabil dan tinggi dan menghasilkan pemodelan yang paling optimal.

Weka menggunakan dokumen berformat ARFF sebagai masukkan *data training* ataupun sebagai *dataset*. Selain menggunakan data *set* untuk pengujian, pustaka pemrograman Weka dapat menggunakan masukan data uji yang dibuat melalui kode sumber. Dalam pengembangan sistem, Weka digunakan untuk melihat tingkat akurasi dari suatu pemodelan yang dibuat. Contoh pengaplikasian perhitungan akurasi Weka dapat dilihat pada Gambar 2.7.

BAB III
PERANCANGAN PERANGKAT LUNAK

Perancangan merupakan bagian penting dari pembuatan perangkat lunak yang berupa perencanaan-perencanaan secara teknis aplikasi yang akan dibuat. Sehingga, bab ini akan dijelaskan mengenai dasar perancangan perangkat lunak yang akan dibuat dalam tugas akhir ini. Secara khusus akan dibahas mengenai deskripsi umum aplikasi, perancangan proses, alur, serta gambaran implementasi perangkat lunak berupa *WPF desktop aplication* dan tampilan pada peramban Mozilla Firefox.

3.1. Deskripsi Umum Sistem

Perangkat lunak pada yang dibangun pada tugas akhir merupakan pendeteksi *URL* berbahaya pada stream twitter. Perangkat lunak ini diharapkan dapat menganalisa rantai *URL* pada *tweet* serta mendeteksi dari akun yang telah menyebarkan *URL* tersebut dengan cara *spam*. Perangkat lunak yang dibangun terdiri atas tiga bagian yaitu *server* basisdata dengan menggunakan MySQL, *crawler tweet* dan pendeteksi berbasis *C# Windows Form Application*, dan Greasemonkey *add-ons* Mozilla Firefox yang menerima hasil analisa *detector* dan menampilkan hasilnya pada peramban Mozilla Firefox.

Analisa *detector* yang dibangun berbasiskan bahasa pemrograman C# ini akan menerima sekumpulan *tweet* yang mengandung *URL*, dan ditampung hingga *windowsize* yang telah ditentukan. Setelah mencapai *windowsize* tertentu, data di *pop-up* dan kemudian masing-masing *URL* pada *tweet* me-*request* lokasi halaman dari *URL* pada *tweets* yang telah menggunakan layanan *shortening-URL* seperti *http://t.co/*, sehingga membentuk suatu rantai *redirect* yang nantinya akan diproses. Proses pengecekan *URL* tersebut bertujuan untuk menentukan *entrypoint*, yakni kondisi satu *URL* pada serangkaian rantai *URL* tersebut mempunyai kemiripan dengan satu *URL* tertentu pada serangkaian

rantai *URL* pada *tweet* yang lain. Sehingga dari *entrypoint* yang dihasilkan, dibagi menjadi lima fitur, yakni: panjang rantai *redirect* pada *URL*, frekuensi munculnya *entrypoint* pada *URL*, letak dari *entrypoint*, banyaknya beda inisial *URL* pada *entrypoint* yang sama, dan jumlah *landing-page* yang dihasilkan. Dari kelima fitur tersebut akan dilakukan pengecekkan, apabila *URL* yang ditemui pada *tweet* tersebut termasuk berbahya maka akan memenuhi kelima fitur tersebut, dan disimpan kedalam *server* basis data.

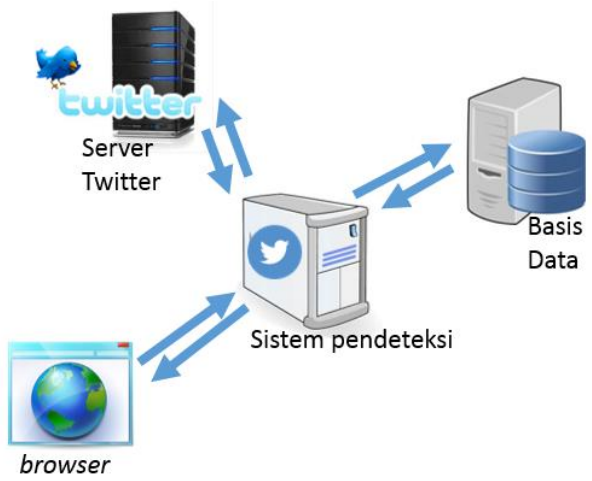
Setelah melakukan pengecekan pada *URL* berbahaya, tercatat sekumpulan *URL* berbahaya dan dapat diketahui akun yang telah mengirimkan *URL* tersebut dan dikategorikan sebagai *attackers*. Dari sekumpulan *attacker* yang tercatat dapat dianalisa dan dibagi menjadi lima fitur untuk ditentukan apakah akun yang mengirimkan *URL* berbahaya tersebut merupakan akun-akun palsu atau bukan, lima fitur yang didapat yakni: jumlah akun yang telah mengirimkan *entrypoint URL* yang sama, simpangan baku dari *follower* dan *following* pada *attacker*, simpangan baku dari rasio *follower-friend*, kemiripan text pada tweet yang telah akun kirim.

Keputusan hasil akhir yang didapat dari fitur-fitur yang dilakukan didapat dengan melakukan penglasifikasian yang menggunakan *decision tree* yang nantinya akan mengetahui akurasi dari sistem pendeteksi tersebut.

Pada pengiriman notifikasi, terdapat dua metode, yakni menggunakan layanan *adds-on* Greasemonkey pada peramban Mozilla Firefox yang nantinya dimasukkan suatu skrip untuk mengubah halaman tampilan dari twitter, sehingga pada saat terindikasi bahwa adanya *URL* yang telah dideteksi berbahaya maka akan mengeluarkan peringatan berupa *pop-up* yang akan menunjukan akun, tweet, dan *URL* yang telah dideteksi berbahaya yang muncul saat di-*refresh* halaman tersebut pada peramban Mozilla Firefox. Sistem *detector* tersebut berjalan secara terus menerus hingga pengecekkan berakhir. Setelah sistem *detector* tersebut berhenti, maka akan mengirimkan notifikasi *email* kepada akun yang telah menggunakan aplikasi tersebut.

3.2. Arsitektur Umum Sistem

Rancangan dapat menjalankan fungsinya, maka alur kerja dari kesatuan aplikasi ini dirancang seperti pada Gambar 3.1.



Gambar 3.1 Rancangan arsitektur sistem

Berdasarkan Gambar 3.1, alur kerja aplikasi ini dijabarkan sebagai berikut:

1. Pengguna (klien) menjalankan aplikasi dengan meminta otorisasi kepada Twitter dan pengguna memasukan token yang telah dikirimi oleh Twitter tersebut ke dalam aplikasi setelah melakukan login.
2. Apabila user baru, maka diarahkan untuk menginstall script.js.
3. *Tweets* yang mengandung *URL* akan dipush ke dalam penampung *tweet* dalam basis data.
4. Setelah mendapatkan sebanyak *n-tweets*. Maka dilakukan pengecekan dimulai dari *URL* yang dicek *redirection*-nya dan dicocokkan masing-masing *URL* pada rantai *redirection* satu dengan yang lain.

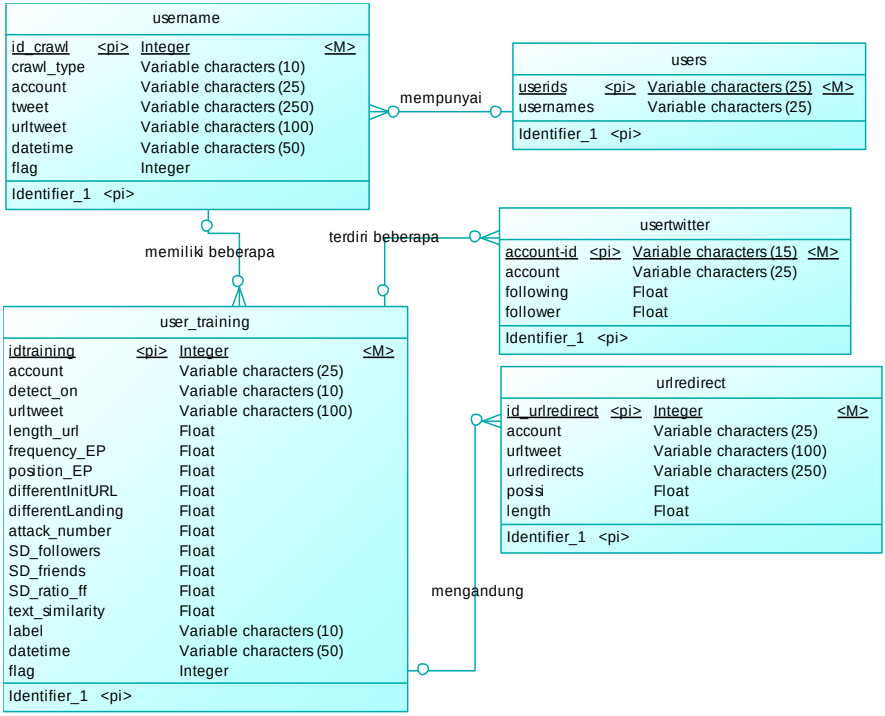
5. Apabila terdapat lebih dari satu *redirect-point URL* yang sama Antara inisial *URL* yang satu dengan yang lain, maka diindikasikan sebuah *entrypoint*.
6. Apabila dalam satu rantai *redirection URL* terdapat lebih dari *entrypoint* maka akan dipilih jumlah frekuensi yang terbesar.
7. Selanjutnya ketika *entrypoint* telah didapatkan maka poin *redirection* setelah *entrypoint* merupakan *Landing URL*. Sehingga dilakukan pengecekan dengan menggunakan dua *user-agent*.
8. Selanjutnya dilakukan pengecekkan *landing URL* oleh dua *user-agent* dan mengunjungi situs tersebut untuk mengetahui *redirect* pada *URL*. Adapun dua *user-agent* tersebut yakni *browser* normal seperti Mozilla Firefox, dan *crawler*.
9. Dari sekian *URL* yang diindikasi berbahaya dikelompokan dan dicari korelasi antara *URL* satu dengan *URL* yang lain, selain itu juga mendapatkan konteks informasi pada *tweet* untuk mengetahui kemiripan suatu akun.
10. Sehingga pengekstrakan fitur didapat berdasarkan dua faktor, yakni berdasarkan korelasi *URL* yang berantai, dan konteks informasi pada *tweet*.
11. Melakukan klasifikasi terhadap atribut-atribut dari fitur yang didapat untuk menentukan akurasi dari status *URL*. Klasifikasi didapat menggunakan metode klasifikasi *decision tree.*
12. Setelah ada pemberitahuan bahwa skrip pada Greasemonkey *add-ons* aktif, maka sistem akan berjalan pada *background-process* dan pengguna dapat memantau *timeline* dan *mention* pada peramban Mozilla Firefox.
13. Skrip pada Greasemonkey *Add-ons* mendapatkan informasi hasil analisa *URL* berbahaya pada *tweet* yang dideteksi pada *timeline* maupun *mention*. Kemudian

memberikan peringatan pada user setelah me-*refresh* halaman Twitter pada peramban Mozilla Firefox.

14. Setelah sistem detektor berhenti, pengguna akan dikirimi notifikasi melalui *email*.

3.3. Perancangan Basis Data

Perancangan basis data menggambarkan struktur tabel yang akan digunakan dalam penyimpanan kebutuhan data sistem pada basis data. Pada Gambar 3.2 menggambarkan *Conceptual Data Model* dari basis data yang digunakan dalam sistem ini.



Gambar 3.2 *Conceptual Data Model* sistem

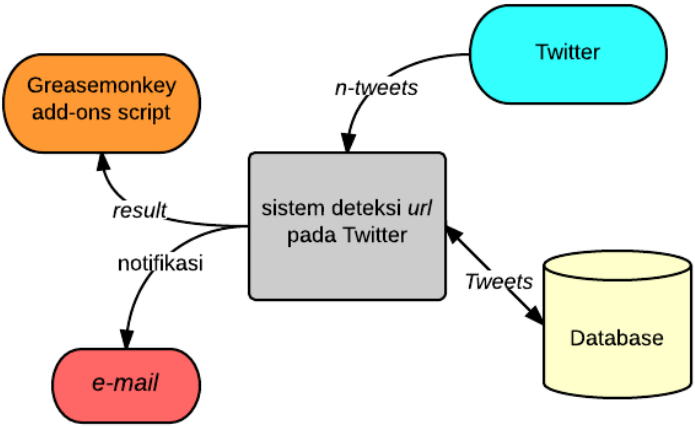
3.4. Perancangan Diagram Alir Data Aplikasi

Pada bagian ini akan dibahas mengenai gambaran aliran data dan fungsionalitas sistem secara umum. Hal ini direpresentasikan berupa diagram konteks dan diagram alir data level 0.

3.4.1. Perancangan Diagram Konteks Aplikasi

Diagram konteks merupakan diagram alir yang menggambarkan sistem secara umum. Semua aktor eksternal serta aliran data masuk dan keluar sistem digambarkan dalam satu diagram, dimana keseluruhan sistem digambarkan dalam satu proses. Konteks diagram aplikasi ini ditunjukkan pada Gambar 3.3.

Sistem ini akan menerima inputan berupa sekumpulan *n-tweets* yang diperoleh dari Twitter setelah melakukan otorisasi. Basis data bertindak mencatat *tweets* baik sebelum diproses maupun setelah diproses. Sistem kemudian melabelkan mana saja *tweet* yang mengandung berbahaya dan mengirimnya ke skrip Greasemonkey. Hasilnya pun juga dikirim kepada pengguna melalui notifikasi.

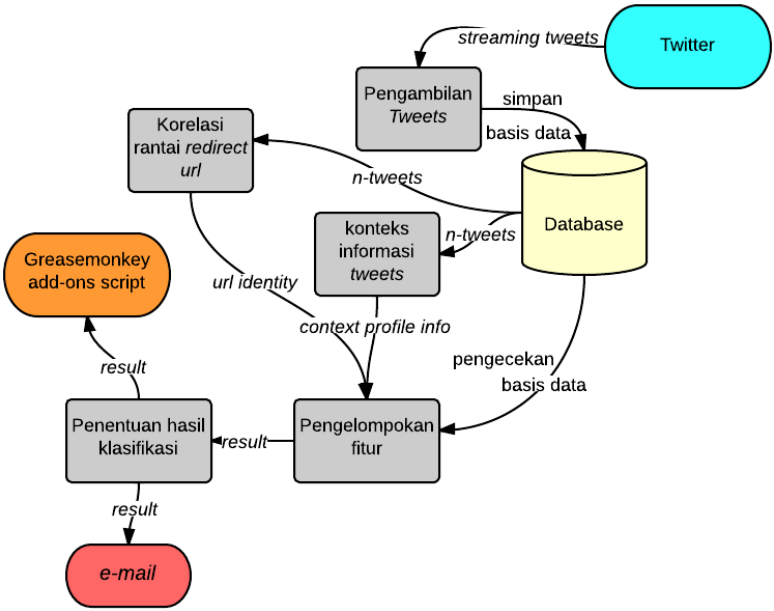


Gambar 3.3 Diagram konteks aplikasi

3.4.2. Diagram Alir Data Level 0 Aplikasi

Diagram alir data level 0 ini merupakan dekomposisi dari proses utama pada diagram konteks. Diagram ini menggambarkan fungsionalitas yang terjadi pada proses di sistem ini. Diagram alir data level 0 ditunjukkan pada Gambar 3.4.

Diagram pada Gambar 3.4 menunjukkan proses-proses yang terjadi pada sistem. Pada tahap awal, sistem melakukan pengumpulan sebanyak *n-tweets* yang didapat dari Twitter API dan di-*push* ke dalam basis data. kemudian dari *n-tweets* tersebut akan diproses untuk mendapatkan korelasi *redirect URL* dan konteks informasi pada *tweet*. Setelah itu, proses dilanjutkan dengan melakukan pengelompokan fitur yang digunakan untuk melakukan klasifikasi. Hasil akhir dari proses ini akan memberikan pelabelan pada masing – masing *tweets*.



Gambar 3.4 Diagram alir data level 0

Dalam proses pengambilan *tweet* dan pengelompokan fitur juga dilakukan penyimpanan basis data sehingga jika terdapat *tweet* yang mengirimkan *URL* yang berbahaya akan disimpan dan dijadikan *pre-defined list* untuk proses selanjutnya. Alamat *URL* yang diakses akan dicek keberadaannya pada basis data yang ada untuk kemudian digunakan sebagai pendukung hasil akhir pelabelan *URL* berbahaya pada *tweet*.

3.5. Diagram Alir Aplikasi

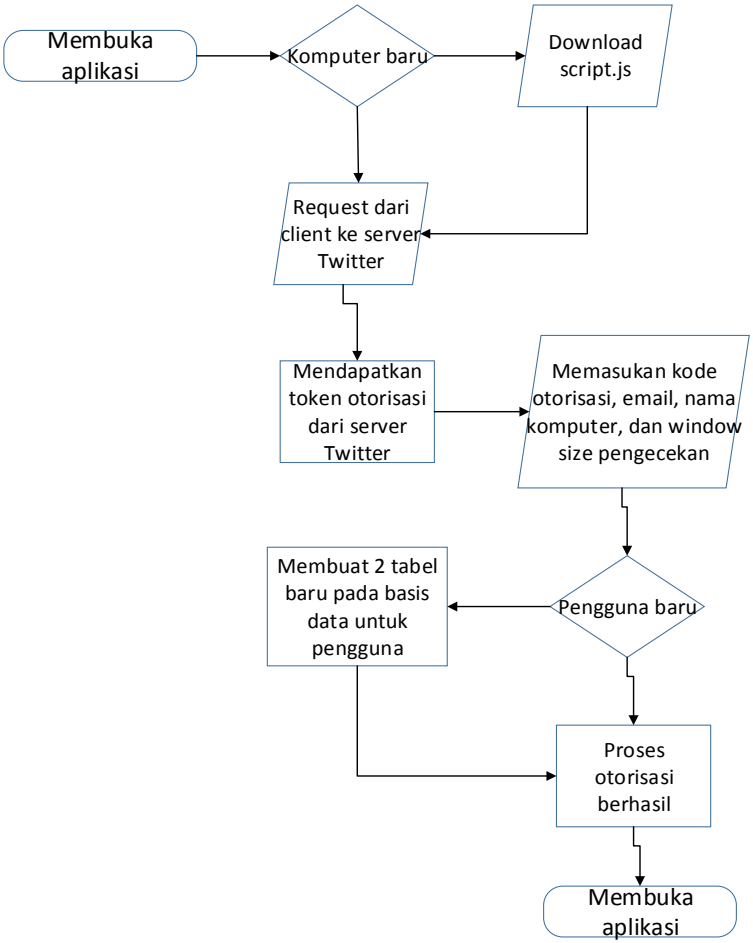
Pada bagian ini akan dijelaskan secara lebih mendetail setiap proses yang terjadi pada sistem. Proses ini akan digambarkan menggunakan diagram alir. Hal ini dimaksudkan untuk mempermudah memahami alur kerja aplikasi pada tugas akhir ini. Diagram yang dibahas pada bagian ini adalah proses yang berjalan pada aplikasi diluar yang dikerjakan oleh pustaka pemrograman dan pustaka. Berikut merupakan masing–masing diagram alir proses yang terdapat pada aplikasi dan akan dijabarkan pada sub-bab selanjutnya.

3.5.1. Diagram Alir Proses Pengambilan *Tweet*

Proses awal yang dilakukan oleh sistem ini ialah melakukan pengambilan *tweet*. Akan tetapi untuk melakukan proses pengambilan *tweet* tersebut diperlukan akses *token* otorisasi yang dikirimkan oleh server Twitter dengan meminta pengguna melakukan *login* pada akun Twitter pengguna. akses *token* otorisasi tersebut berfungsi memberi izin aplikasi mengakses segala informasi dan data pengguna yang dibutuhkan oleh sistem pada *server* Twitter. Untuk proses pengambilan *tweet* dapat dilihat pada Gambar 3.5.

Seperti yang telah dipaparkan pada Gambar 3.5, setelah mendapatkan hak akses *token* otorisasi tersebut, pengguna diminta untu memasukkan *token* tersebut kedalam sistem. Selain itu pengguna diminta untuk memasukan alamat *email* untuk pengiriman notifikasi, nama komputer untuk pengecekan

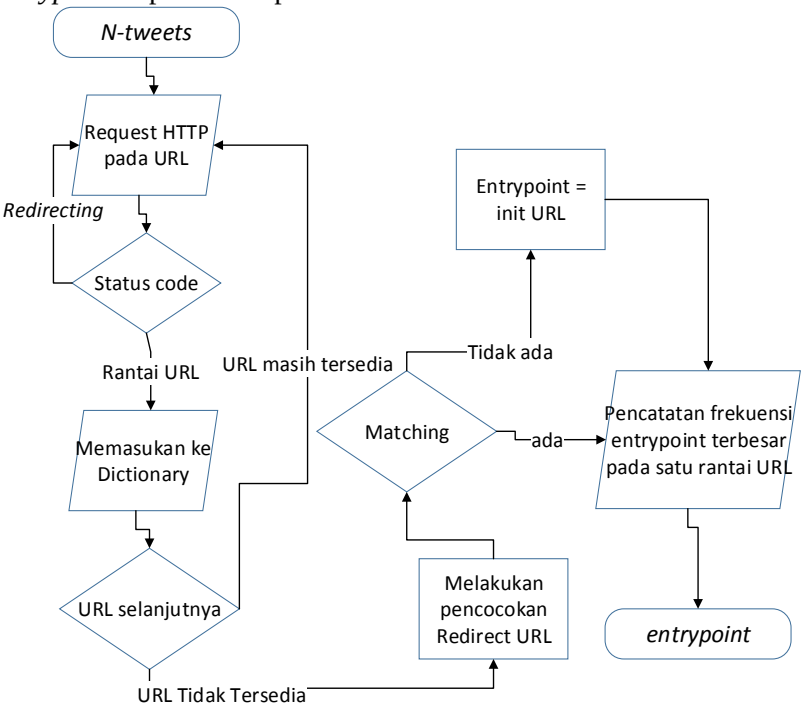
pelabelan pada peramban, serta *window size* untuk melakukan banyaknya *tweet* yang diproses dan dilakukan pengecekan.



Gambar 3.5 Diagram alir proses pengambilan *tweet*

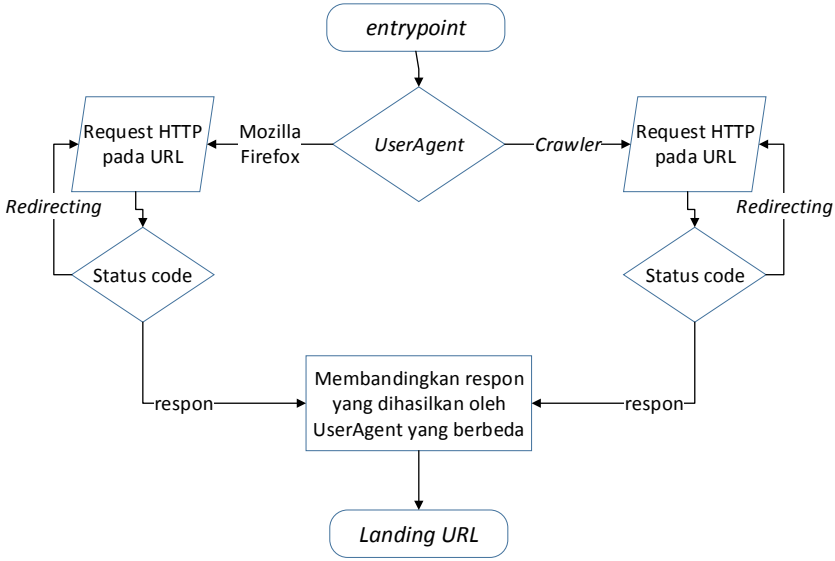
3.5.2. Diagram Alir Proses Ekstraksi Korelasi Rantai *Redirect URL*

Setelah melakukan pengambilan data *tweet* pada *server* Twitter sebanyak *n-tweets* sistem melakukan proses ekstraksi korelasi rantai *redirect URL* pada setiap *tweet*-nya. Hal ini diperlukan untuk melakukan pengambilan fitur pada langkah selanjutnya. Proses ekstraksi ini dibagi menjadi dua bagian, yakni proses pencarian *entrypoint* dan proses pencarian *landing-URL* pada *entrypoint* yang dihasilkan. Untuk proses pencarian *entrypoint* dapat dilihat pada Gambar 3.5.



Gambar 3.5 Diagram pencarian *entrypoint URL*

Dari Gambar 3.6 terlihat bahwa *URL* pada *tweet* akan dilihat rantai *redirect* dengan melakukan *request* HTTP berulang – ulang hingga menuju pada *domain* yang diinginkan pada kondisi kode status 200. Kemudian data ditampung dan dijadikan *dictionary* sebagai penanda urutan pada sekumpulan *tweets*. Kemudian melakukan pencocokan *URL* pada rantai *URL*. Apabila didalam satu rantai *URL* terindikasi adanya *URL* yang sama dengan *URL* pada rantai *URL* yang lain, maka dijadikan *entrypoint*, dan dihitung jumlah frekuensi kemunculan *URL* tersebut. Apabila terdapat lebih dari satu *entrypoint* yang sama pada *tweet* maka dipilih *entrypoint* dengan jumlah frekuensi terbesar. Kemudian setelah melakukan proses pencarian *entrypoint*, sistem melakukan pencarian *landing-URL* pada masing-masing *entrypoint* yang dihasilkan.

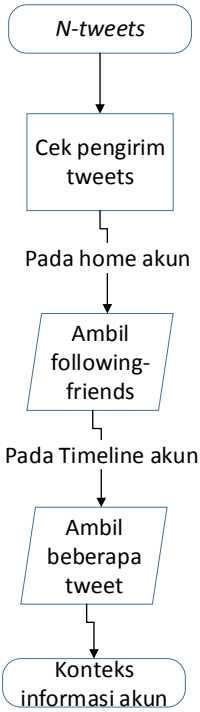


Gambar 3.6 Diagram alir pencarian *Landing URL*

Seperti yang telah dijelaskan pada Gambar 3.6, proses pencarian *landing-URL* dilakukan dengan melakukan *request*

HTTP untuk mendapatkan respon yang dihasilkan. Dalam melakukan *request* HTTP, sistem menggunakan *UserAgent* yang berbeda, yakni Mozilla Firefox sebagai peramban yang normal, dan juga *crawler*. Apabila pada *entrypoint* tersebut terdapat perbedaan *landing-URL*, maka didapatkan dicatat untuk fitur selanjutnya pada tahapan ekstraksi fitur.

3.5.2. Diagram Alir Proses Ekstraksi Konteks Informasi *Tweets*



Gambar 3.7 Diagram alir konteks informasi *tweets*

Diagram alir proses ini dapat dilihat pada Gambar 3.7. Sama seperti ekstrasi korelasi rantai *redirect URL*, Konteks informasi *tweet* pada *URL* yang mempunyai kesamaan berbahaya tersebut

diperlukan untuk proses pengelompokan fitur di tahap selanjutnya. Namun berbeda pada *redirect URL identity*, informasi *tweet* diambil dengan mendapatkan entitas dari *n-tweets* yang didapat dari Twitter API dan sudah dicatat oleh basis data. berdasarkan parameter yang nantinya dapat dijadikan beberapa fitur.

Dari Gambar 3.7, terlihat proses yang dilakukan dalam ekstraksi konteks informasi pada *tweet*. Hal ini diproses dengan mengambil informasi pada pengirim yang telah mengirimi *tweet* yang mengandung *URL* tersebut, dimana mengambil informasi *followings* dan *followers* akun pengirim kemudian mengambil isi *timeline* dari akun.

3.5.3. Diagram Alir Proses Ekstraksi dan Klasifikasi Fitur

Pada proses ini akan dijabarkan proses pengelompokan klasifikasi fitur dengan hasil akhir pelabelan dari *tweets* pada *timeline* dan *mention* pengguna. Terdapat sepuluh fitur yang digunakan pada proses ini dan akan dijabarkan satu per satu untuk setiap prosesnya. Setiap fiturnya akan dilakukannya normalisasi [12] antara nol dan satu.

3.5.3.1. Pengecekan Panjang Rantai dari suatu *URL*

Pada proses ini dilakukan pengecekan dan mencatat panjang rantai dari suatu *URL* dimana masing–masing *URL* pada *tweet* mengalami *redirect* dan membentuk suatu rantai. Dari sini dicek panjang dari rantai *URL* tersebut. hal yang mendasari bahwa *URL* yang mencurigakan adalah *URL* tersebut mempunyai rantai *redirect* yang lebih panjang dari *URL* biasanya.

Contoh kasus dari *redirect URL* ialah *http://t.co/sadkjas* ke *http://bit.ly/1jadsks* hingga halaman yang dituju. Untuk melakukan normalisasi pada panjang *redirect*, ditentukan batas atas nilai dari panjang *redirect* yakni 7. Karena berdasarkan analisa sebelum-sebelumnya, sebagaian besar didapat kurang dari tujuh pengalihan *URL* dalam satu rantai *redirect URL*. Sehingga normalisasinya $\mathbf{n}(\boldsymbol{l}, \mathbf{7})\ /\mathbf{7}$ [1].

3.5.3.2. Pengecekan Frekuensi dari *Entrypoint URL*

Pada bagian ini, dilakukan pengecekan frekuensi dari *entrypoint* pada *URL* di setiap *tweet* pada *n-tweets*. Seperti yang dijelaskan pada sub-bab 3.5.1., dimana pencarian *entrypoint* dilakukan dengan mencocokkan *URL* satu dengan *URL* lain. Apabila dalam *n-tweets* tersebut terdapat lebih dari satu *URL* yang sama maka dicatat frekuensi kemunculannya. Apabila dalam satu *tweet* terdapat lebih dari satu *entrypoint*, maka dipilih frekuensi *entrypoint* terbesar. Apabila dalam satu *tweet* tidak ditemukannya *entrypoint* maka inisial *URL* dijadikan *entrypoint* dengan frekuensi satu. Jika *n-tweets* yang diambil dalam sekali pengecekan adalah $\boldsymbol{w}$ dan frekuensi *entrypoint* yang terjadi disetiap *tweet*nya adalah $\boldsymbol{n}$ maka normalisasinya ialah $\boldsymbol{n/w}$ [1].

3.5.3.3. Pengecekan Letak dari *Entrypoint URL*

Biasanya *entrypoint URL* yang mencurigakan tidak terletak pada akhir rantai *redirect URL* karena *URL* tersebut mengalami pengalihan kondisional untuk mengarahkan pengunjung pada *landing URL* yang berbeda. Jika posisi *entrypoint* disimbolkan $\boldsymbol{p}$ dan panjang rantai *redirect URL* dinotasikan $\boldsymbol{l}$, maka normalisasi fitur ini menjadi $\boldsymbol{p/l}$ [1].

3.5.3.4. Pengecekan Jumlah Inisial *URL* Pada *Entrypoint* yang Sama

Inisial *URL* yang dimaksud adalah *URL* yang terlihat pada pengguna. Satu Akun dengan akun yang lain ketika mendistribusikan *tweet*nya yang mengandung *URL*, akan meng-*generate shortening URL* yang berbeda–beda meskipun mengarah ke *entrypoint* yang sama, sehingga dengan sistem ini melakukan proses pengecekan dan menghitung jumlah inisial *URL* yang berbeda ketika mengalami pengalihan ke *entrypoint* yang sama. Jika banyaknya inisial *URL* yang mempunyai *entrypoint* yang muncul sebanyak $\boldsymbol{n}$ dinotasikan $\boldsymbol{i}$, maka dinormalisasikan menjadi $\boldsymbol{i/n}$ [1].

3.5.3.5. Pengecekan Jumlah *Landing Page* Pada *Entrypoint* yang Sama

Pada proses ini, sistem melalukan pengecekan jumlah *landing page* pada setiap *entrypoint URL*. Apabila terindikasi bahwa pada suatu tautan mengalami perbedaan *landing page* maka disimpulkan bahwa *landing page* >1. Jika *entrypoint* yang muncul sebanyak $\boldsymbol{n}$ melakukan perbedaan pengalihan *landing page* sebanyak $\boldsymbol{\lambda}$, maka normalisasinya adalah $\boldsymbol{\lambda/n}$ [1].

3.5.3.6. Pengecekan Jumlah Pengirim *URL* Pada *Tweet* Dengan *Entrypoint* yang Sama

Pada proses ini, sistem melakukan pengecekan jumlah akun yang mengirimkan *URL* dengan *entrypoint* yang sama. Karena banyak penyerang yang membuat akun – akun palsu untuk mendistribusikan *tweet*-nya. jumlah pengirim dihitung berdasarkan seberapa banyak frekuensi *entrypoint* yang muncul, sehingga ketika terdapat lebih dari satu nilai frekuensi *entrypoint* akan tetapi diketahui bahwa *entrypoint URL* tersebut berasal dari pengirim yang sama maka hanya diambil salah satunya. Untuk menghindari pengambilan jumlah pengirim yang lebih maka dilakukan normalisasi. Apabila jumlah akun twitter yang telah mengirimkan *entrypoint* yang muncul sebanyak $\boldsymbol{n}$ kali disimbolkan sebagai $\boldsymbol{\alpha}$, maka dinormalisasikan menjadi $\boldsymbol{\alpha/n}$ [1].

3.5.3.7. Pengecekan Simpangan Baku Pada *Followers* Pengirim

Akun-akun palsu yang dibuat oleh penyerang biasanya mempunyai angka kemiripan yang besar. Hal ini mempertimbangkan pada status akun seperti *followers* dan *friends* dari akun tersebut. Pada proses ini, *followers* dari akun tersebut akan dihitung simpangan baku *followers* antara akun satu dengan akun yang lain. Simpangan baku tersebut digunakan untuk mengecek kemiripan dari angka *followers* akun satu dengan yang lainnya. Diketahui kemungkinan akun-akun palsu yang didapat

dalam melakukan *follow* mencapai kurang lebih 200 pengguna sehingga fitur ini dinormalisasikan seperti pada Persamaan (3.1) [1].

$$\boldsymbol{min}\left(\frac{\boldsymbol{std}(\#\boldsymbol{followers})}{\boldsymbol{200}\sqrt{\boldsymbol{n}}}, \boldsymbol{1}\right) \quad \textbf{(3.1)}$$

3.5.3.8. Pengecekan Simpangan Baku Pada *Friends* Pengirim

Seperti yang dijelaskan pada sub-bab 3.4.3.7, pada proses ini juga mengecek *friends* dari masing-masing pengirim dan menghitungnya dengan menggunakan simpangan baku. Hal ini diperhitungkan untuk mengecek dari angka *friends* akun satu dengan yang lainnya. Semakin mirip angka *friends* dari masing-masing akun yang mempunyai *entrypoint URL*. Diketahui kemungkinan akun-akun palsu yang didapat dalam melakukan *follow* mencapai kurang lebih 200 pengguna sehingga fitur ini dinormalisasikan seperti pada Persamaan (3.2) [1].

$$\boldsymbol{min}\left(\frac{\boldsymbol{std}(\#\boldsymbol{friends})}{\boldsymbol{200}\sqrt{\boldsymbol{n}}}, \boldsymbol{1}\right) \quad \textbf{(3.2)}$$

3.5.3.9. Pengecekan Simpangan Baku Pada Rasio Dari *Followers-Friends* Pengirim

Selain melakukan pengecekan terhadap simpangan baku *followers* dan *friends* juga dilakukan pengecekan kemiripan perbandingan antara *follower* dan *following*. Hal ini juga menghitung angka kemiripan dari akun satu dengan akun yang lain. Akun – akun palsu yang dibuat oleh penyerang biasanya mempunyai angka *friends* yang lebih besar dengan *followers*. Sehingga menggunakan perbandingan Antara *friends* dan *followers* yang biasanya mirip antara akun satu dengan akun yang lain. Apabila untuk mendapatkan rasio dari pada *followers-friends* didapatkan seperti yang ditunjukan pada Persamaan (3.3) [1].

$$\frac{\boldsymbol{min}(\#\boldsymbol{followers},\#\boldsymbol{following})}{\boldsymbol{max}(\#\boldsymbol{followers},\#\boldsymbol{following})} \quad \textbf{(3.3)}$$

Dimana notasi {#} merupakan sekumpulan, maka normalisasinya pada fitur ini ditunjukan pada Persamaan (3.4) [1].

$$\boldsymbol{min}\left(\frac{\boldsymbol{std}(\boldsymbol{rasio\ pada\ followers-friends})}{\sqrt{\boldsymbol{n}}}, \mathbf{1}\right) \quad \textbf{(3.4)}$$

3.5.3.10. Pengecekan Kemiripan Teks *Tweets* Pada Pengirim

Pada tahapan ini, akan dilakukan pengecekan terhadap isi *timeline* dari pengirim. Dari *tweets* yang ada di *timeline* maupun *mention* pengguna, akan dicocokkan dengan isi *timeline* dari pengirim tersebut dengan menggunakan *Jaccard Index* [13]. Biasanya, akun – akun palsu tersebut mendistribusikan *tweet*-nya dengan cara *spam* sehingga besar kemungkinan bahwa kemiripan teks pada *timeline* akun tersebut sama. Hal ini dinormalisasikan seperti yang ditunjukan pada Persamaan (3.5) [1].

$$\sum_{\boldsymbol{t,u} \in \boldsymbol{pasangan\ teks\ pada\ tweets}} \frac{\mathcal{J}(\boldsymbol{t,u})}{|\boldsymbol{pasangan\ teks\ pada\ tweets}|} \quad \textbf{(3.5)}$$

Dimana *jaccard index* ditunjukan pada Persamaan (3.6) [1].

$$\mathcal{J}(\boldsymbol{t,u}) = \frac{|t \cap u|}{|t \cup u|} \quad \textbf{(3.6)}$$

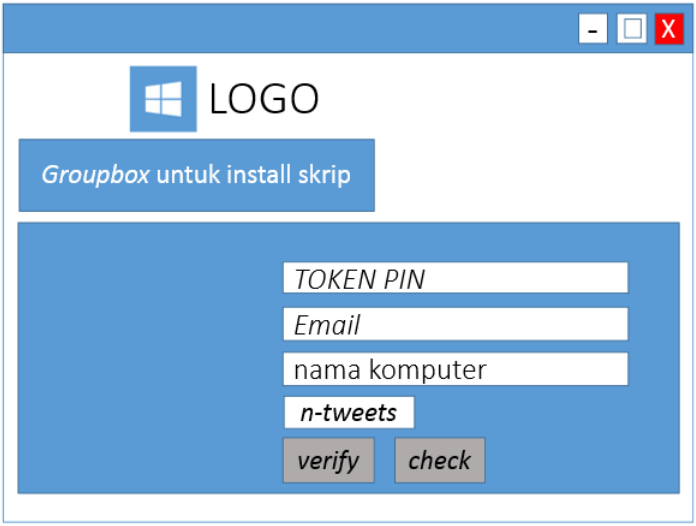
3.6. Rancangan Antarmuka

Terdapat dua bagian aplikasi yang berhubungan langsung dengan pengguna pada tugas akhir ini, yakni aplikasi pada *desktop* dan juga pada peramban Mozilla Firefox. Untuk itu diperlukan perancangan antarmuka keduanya.

3.6.1. Tampilan Pada Layar *Desktop*

Pada Gambar 3.8 menunjukan bagaimana tampilan pada layar *desktop*. Untuk menjalankan sistem pendeteksi *URL* berbahaya tersebut, pertama kali yang harus dilakukan ialah berhubungan langsung dengan layar *desktop*. Apabila aplikasi tersebut akan

dijalankan pada komputer dan peramban yang belum pernah dipakai sebelumnya, maka terdapat *groupbox* untuk menginstall skrip. Pada *groupbox* yang kedua, pengguna mengisikan *token* PIN otorisasi yang dikirimi oleh Twitter, email pengguna, nama komputer, dan tombol *start* untuk memulai proses pendeteksian.



Gambar 3.8 Rancangan antarmuka tampilan awal

3.6.2. Tampilan Pada Peramban Mozilla Firefox

Gambar 3.9 Rancangan antarmuka menu pencarian

Rancangan antarmuka pada Gambar 3.9 menunjukkan bagaimana skrip pada Greasemonkey *add-ons* memberikan peringatan dengan menampilkan peringatan bila terdapat *tweet* yang mengandung *URL* berbahaya dari halaman Twitter yang dikunjungi, yakni pada *timeline* dan *mention.*

BAB IV
IMPLEMENTASI

Setelah melalui proses analisis dan perancangan perangkat lunak, dilakukan implementasi sistem. Bab ini membahas tentang proses implementasi pada sistem. Tahapan implementasi meliputi implementasi program dalam *pseudocode* dan antarmuka yang telah dibahas pada BAB III.

4.1. Lingkungan Implementasi

Dalam proses implementasi sistem ini, digunakan beberapa perangkat pendukung untuk memenuhi kebutuhan sistem yang terdiri dari perangkat keras dan perangkat lunak.

4.1.1. Lingkungan Implementasi Perangkat Keras

Dalam proses implementasi, spesifikasi perangkat keras yang digunakan untuk pembangunan aplikasi tugas akhir ini adalah menggunakan Notebook COMPAQ Presario CQ42, Intel Core i5 2430M 2.4 Ghz, 4 GB RAM.

4.1.2. Lingkungan Implementasi Perangkat Lunak

Beberapa perangkat lunak yang digunakan untuk kebutuhan implementasi sistem diantaranya adalah:

- Sistem operasi Windows 8 *Enterprise* 64-bit.
- Microsoft Visual Studio 2012 Ultimate untuk pembangunan aplikasi deteksi *URL* pada Twitter.
- Microsoft Office Word 2013 untuk melakukan dokumentasi program dan menyelesaikan buku.
- Weka 3.6.10 untuk melakukan proses uji coba pemodelan data pada sistem.
- MySQL Server 5.5.27 sebagai basis data.
- MySQL Connector Net 6.8.3 sebagai penghubung basis data dengan aplikasi pengembang Microsoft Visual Studio.

- XAMPP sebagai *server* yang berjalan pada *localhost* dan digunakan untuk menjalankan basis data MySQL.
- Microsoft Visio Professional 2013 untuk membuat rancangan antarmuka dan *flowchart* diagram alur.
- PowerDesigner versi 15.0 untuk membuat *Conceptual Data Model* dan *Pyshical Data Model*.
- Mozilla Firefox v.30 dan Greasemonkey *add-ons* sebagai uji coba notifikasi peringatan pada Twitter.

Selain perangkat lunak yang telah disebutkan di atas, aplikasi pada *client* juga menggunakan tiga *library* pendukung dalam men yang sudah ada, yaitu:

- *TweetSharp-Unofficial* versi 2.3.1.2 sebagai penunjang pengembangan aplikasi dalam mengambil informasi pengguna pada *server* Twitter.
- Google *Mail Client* digunakan sebagai penunjang aplikasi dalam mengirimkan *e-mail.*

4.2. Implementasi Perangkat Lunak

Pada sub bab ini akan dibahas mengenai implementasi dari proses digunakan untuk pembangunan aplikasi tugas akhir. Implementasi dilakukan pada sisi pembangunan detektor *URL* berbahaya pada Twitter menggunakan aplikasi *desktop,* ekstraksi dan klasifikasi fitur, serta Greasemonkey *add-ons* pada Mozilla Firefox.

Berikut ini adalah penjelasan implementasi dari proses utama yang terjadi pada sistem. Proses-proses implementasi dijelaskan sesuai dengan urutan berjalannya sistem. Setiap bagian akan dijelaskan lebih lanjut pada masing-masing subbab.

4.2.1. Implementasi Deteksi *URL* pada *Tweet*

Pada sisi *server* tahapan awal yang dilakukan adalah melakukan *crawling tweet* yang mengandung *URL* pada Twitter dengan menggunakan pustaka pemrograman *TweetSharp-Unofficial* versi 2.3.1.2., dan ditampung kedalam basis data

sebanyak *n-tweets*. Dalam melakukan pendeteksian *URL* berbahaya, ada parameter yang dilakukan oleh sistem, yakni: proses pengambilan *tweet*, proses ekstraksi korelasi rantai pengalihan *URL* pada *tweets*, proses ekstraksi konteks informasi *tweets*, proses ekstraksi fitur, dan proses klasifikasi fitur.

4.2.1.1. Implementasi Proses Pengambilan *Tweet*

```
    Kirim request ke server Twitter
2   If request = true
3       Membuka default browser
4       GET kode pengguna
5       Masukan akses token
6       Masukan email pengguna
7       Masukan nama komputer
8       Masukan window size tweet pengecekan
9       If kode otorisasi = true
10          GET username pengguna
11          FIND username di dalam basis data
12          If ID ditemukan
13              crawlTweet()  //proses crawling
14          Else
15              Buat username pengguna
                Buat user training //untuk pengecekan
17              checkTweet() //proses crawling
18      Else
19          Kirim request ke server Twitter
20  Else
21      Keluar dari aplikasi
```

Gambar 4.1 *Pseudocode* proses *login*

Pada proses pengambilan *tweet*, proses awal yang dilakukan oleh aplikasi yakni meminta hak otorisasi pada server Twitter dengan melakukan *login* pada peramban sehingga oleh Twitter mengirimkan akses Token untuk dapat diakses melalui aplikasi. Proses *login* melakukan *query* pada basis data untuk mengecek ketersediaan pengguna. Apabila pengguna ditemukan maka data tersebut siap untuk melakukan pengambilan *tweet*, namun apabila belum tersedia, maka otomatis akan menyimpan data pengguna. Proses yang berjalan di dalam *login* dapat dilihat pada Gambar 4.1.

Kode sumber dari proses ini terdapat pada Lampiran A yang berisi proses *login* secara lengkap.

Setelah melakukan proses *login*, terdapat proses pengambilan *tweet*. Saat melakukan pengambilan *tweet*, data satu persatu ditampung kedalam basis data, selain itu setiap *tweet* akan dicek akun pengirim untuk mengambil *followers-friends*, dan diakses rantai pengalihan *URL* dengan menggunakan *HttpWebRequest* dan ditampung kedalam basis data, sehingga dapat dibutuhkan saat proses pengecekan berlangsung. Untuk implementasi proses pengambilan *tweet* ini dapat dilihat pada Gambar 4.2. Kode sumber dari proses ini terdapat pada Lampiran B.

```
1   Deklarasi mentions = 100
2   Deklarasi tweets = 100
3   COUNT data table username //pengecekan
4   If data = 0
5   Foreach (m in mentions)
6       Simpan m ke username
7       Simpan akunpengirim()
8       Simpan RedirectURL()
9   Foreach (t in tweets)
10      Simpan t ke username
11      Simpan akunpengirim()
12      Simpan RedirectURL()
13  Else
14      Foreach (m in mentions)
15          If(tanggal m > tanggal max mention)
16          Simpan m ke username
17          Simpan akunpengirim()
18          Simpan RedirectURL()
19      Foreach (t in tweets)
20          If (tanggal t > tanggal max tweet)
21          Simpan t ke username
22          Simpan akun pengirim()
23          Simpan RedirectURL()
```

Gambar 4.2 *Pseudocode* proses pengambilan *tweet*

4.2.1.1. Implementasi Proses Ekstraksi Korelasi Rantai *Redirect URL*

Dalam proses ekstraksi korelasi rantai *redirect URL*, sistem akan melakukan pengecekan *tweet* sebanyak *windows size* yang

telah ditentukan. Pengecekan dimulai dengan mencari *entrypoint URL*, dimana terdapat lebih dari satu *URL* yang sama antara satu *URL* pada rantai *URL* yang satu dengan satu *URL* yang lain. Selain melakukan *pop-up* data *tweet* pada basis data, juga melakukan *pop-up* rantai *redirect URL* yang dimasukkan kedalam *dictionary* yang nantinya akan dicek korelasi antara rantai *redirect URL* satu dengan yang lain guna mendapatkan *entrypoint URL* yang dihasilkan. Implementasi proses ini dapat dilihat pada Gambar 4.3. Kode sumber dari proses ini terdapat pada Lampiran C.

```
1   Ambil inisial_url pada tabel usertraining
2   Inisialisasi Dictionary<int, list<string> redir>
3   Foreach(s in inisial_url)
4        Masukan ke dictionary rantai redirect URL
5        Indeks++
6   Foreach key in dictionary
7     Inisialisasi entrypoint
8     For value dictionary
9        Flag=1
10       Mencocokan list(entrypoint,freq) yg sudah ada
11       If value = list entrypoint
12         Set freq
13         Flag = freq
14         Continue
15       For i = key+1
16          Foreach URL2 in dictionary[i]
17            IF value = URL2
18               Freq++
19       SET entrypoint = value[0] //set initial URL
20       If flag < freq
21          SET Entrypoint = current value
22          Masukkan list(entrypoint,freq)
23    Masukan list EP_point<> //list entrypoint
24    Masukan list EP_freq<> //list frekuensi
```

Gambar 4.3 *Pseudocode* proses pencarian *entrypoint URL*

Pada pengambilan *entrypoint* dijelaskan bahwa data diambil dari masing – masing *tweet* dan korelasi rantai *redirect URL* yang telah ditampung oleh basis data. *Output* yang dihasilkan yakni berupa hasil *entrypoint* yang didapat dan frekuensi kemunculan pada *n-*

tweets. Apabila tidak ditemukannya kesamaan *URL* pada rantai *URL* yang satu dengan yang lain, maka *entrypoint* diset sebagai inisial *URL*.

4.2.1.2. Implementasi Proses Ekstraksi Konteks Informasi *Tweet* Pada Pengirim

Pada proses ekstraksi konteks informasi *tweet* pada pengirim ini dilakukan untuk mengetahui status pengirim yang telah disimpan oleh basis data sebelumnya. Masing – masing data dicek status *followers* dan *friends* pada Twitter, selain itu mengambil beberapa *tweets* pada *timeline* akun pengirim yang telah mengirimi *tweet* yang mengandung *URL* untuk dicek kemiripan teksnya. Implementasi pada proses ini dapat dilihat pada Gambar 4.4. Kode sumber dari proses ini terdapat pada Lampiran D.

```
1   Deklarasi mention = 100
2   Deklarasi tweet = 100
3   COUNT data table username //pengecekan
4   If data = 0
5   Foreach (m in mentions)
6       GET followers dan friends pengirim
7       GET tweet pada timeline pengirim
8   Foreach (t in tweets)
9       GET followers dan friends pengirim
10      GET tweet pada timeline pengirim
11  Else
12      Foreach (m in mentions)
13          GET followers dan friends pengirim
14          GET tweet pada timeline pengirim
15      Foreach (t in tweets)
16          GET followers dan friends pengirim
17          GET tweet pada timeline pengirim
```

Gambar 4.4 *Pseudocode* proses ekstraksi konteks informasi *tweet* pada pengirim

4.2.2. Implementasi Proses Ekstraksi Fitur

Pada subbab-subbab berikut akan dijelaskan implementasi proses ekstraksi dan klasifikasi fitur. Terdapat sepuluh fitur yang dihasilkan. Penjelasan implementasi ditulis dalam bentuk *pseudocode*.

4.2.2.1. Implementasi Fitur Pengambilan Nilai Panjang Rantai *Redirect URL*

Pada setiap *URL* yang diambil dari *tweet* mengalami pengalihan *URL* yang membentuk suatu rantai *redirect URL*. Dalam proses ini dilakukan pengecekan panjang rantai *redirect URL* untuk mengetahui apakah *URL* yang terkandung dalam *tweet* tersebut termasuk berbahaya atau tidak. pada pengambilan nilai panjang rantai *redirect URL* dilakukan normalisasi, dan diasumsikan batas atas dari panjang *redirect* suatu *URL* adalah 7. Implementasi pada proses ini dapat dilihat pada Gambar 4.5. Kode sumber lengkap dapat dilihat pada Lampiran E.

```
1  Foreach url in entrypoint
2     SELECT initURL, length pada tabel urlredirect
   dengan entrypoint tertentu
3     Jika length >= 7
4        Length = 7
5     Normalisasi: length / 7
6     Memasukkan parameter initURL ke dalam listURL
7     Memasukkan normalisasi ke dalam listLength
8  For isi listURL
9     UPDATE usertraining SET length=normalisasi
```

Gambar 4.5 *Pseudocode* proses pencatatan panjang *redirect URL*

4.2.2.2. Implementasi Fitur Pengambilan Nilai Frekuensi *Entrypoint URL*

Pada saat proses pencarian *entrypoint URL* berlangsung, juga dilakukan pencatatan frekuensi kemunculan *entrypoint* pada *windowsize* sejumlah *n-tweets*. Sehingga setiap barisnya mempunyai masing-masing *entrypoint* dan frekuensi *entrypoint*.

```
1  FOR mengambil data entrypoint
2      SELECT initURL dengan entrypoint tertentu
3      Memasukkan initURL ke dalam listURL
4      Normalisasi: Freq_EP / windowSize
5      Foreach url in listURL
6          UPDATE usertraining SET freq = normalisasi
```

Gambar 4.6 *Pseudocode* proses pencatatan frekuensi *entrypoint*

Pada proses pengambilan nilai frekuensi *entrypoint URL* ini, dilakukan normalisasi seperti yang telah dijelaskan pada bab III. Implementasi pada proses ini dapat dilihat pada Gambar 4.6. Kode sumber lengkap dapat dilihat pada Lampiran F.

4.2.2.3. Implementasi Fitur Pengambilan Nilai Letak *Entrypoint URL*

Setiap *tweet* yang dicek mempunyai letak *entrypoint* yang berbeda-beda pada rantai *redirect URL*. Sehingga sistem mencatat letak dari setiap *entrypoint* yang muncul pada sekumpulan *tweet*. Pada proses pengambilan nilai letak *entrypoint URL* dilakukan normalisasi seperti yang dijelaskan pada bab III. Gambar 4.7 merupakan *pseudocode* proses pencatatan letak *entrypoint URL*. Kode sumber lengkap dapat dilihat pada Lampiran G.

```
1  Foreach url in entrypoint
2     SELECT initURL, length, posisi pada tabel
   urlredirect dengan entrypoint tertentu
3     Normalisasi: posisi / length
4     Memasukkan parameter initURL ke dalam listURL
5     Memasukkan normalisasi ke dalam listLength
6  For isi listURL
7     UPDATE usertraining SET posisi=normalisasi
```

Gambar 4.7 *Pseudocode* proses pencatatan letak *entrypoint URL*

4.2.2.4. Implementasi Fitur Pengambilan Nilai Jumlah Inisial *URL* Pada *Entrypoint URL*

```
1  For mengambil data entrypoint
2      SELECT initURL dengan entrypoint tertentu
3      Memasukkan initURL ke dalam listURL
4      Normalisasi: listURL.count / Freq_EP
5      SELECT initURL dengan entrypoint tertentu
6      Foreach url in listURL
7          UPDATE usertraining SET freq = normalisasi
```

Gambar 4.8 *Pseudocode* proses pencatatan jumlah inisial *URL* pada *entrypoint URL* yang sama

Implementasi *psuedocode* proses ini ditunjukan pada Gambar 4.8. Pada proses ini dilakukan pengambilan nilai jumlah inisial

URL yang terkandung pada *tweets*, dimana mempunyai *entrypoint URL* yang sama dengan *URL* yang lain. Seperti hal nya pada sub bab yang sebelumnya, untuk mengambil nilai perbedaan inisial *URL* pada *entrypoint URL* yang sama dilakukan normalisasi. Kode sumber lengkap dapat dilihat pada Lampiran H.

4.2.2.5. Implementasi Fitur Pengambilan Nilai Perbedaan *Landing URL*

Setelah mendapatkan *entrypoint URL* pada masing-masing *tweets*, juga dilakukan pengecekan perbedaan *landing URL*. Apabila terdapat perbedaan *landing URL* ketika dicek dengan menggunakan *UserAgent* peramban normal dan juga *crawler*. Proses ini juga dilakukan normalisasi. Implementasi *psuedocode* proses ini ditunjukan pada Gambar 4.9. Kode sumber lengkap dapat dilihat pada Lampiran I.

```
1   For mengambil data entrypoint
2       Dicek dengan menggunakan UserAgent1
3       Simpan ke list1 redirectnya
4       Dicek dengan menggunakan UserAgent2
5       Simpan ke list2 redirectnya
6       For cek isi redirect
7          Jika list1 = list2
8              continue
9          Jika list1 != list2
10              Catat perbedaan landing
11      Normalisasi: landing / freq_EP
12      SELECT initURL WHERE current entrypoint
13      Foreach url in initURL
14     UPDATE usertraining SET diffLanding=normalisasi
```

Gambar 4.9 *Pseudocode* proses pencatatan nilai perbedaan *landing URL*

4.2.2.6. Implementasi Fitur Pengambilan Jumlah Pengirim *Entrypoint URL*

Pada proses ini dilakukan pengambilan jumlah pengirim yang telah mengirimkan *entrypoint URL* yang sama. Apabila terdapat lebih dari satu frekuensi *entrypoint* maka dapat dicurigai mereka mendistribusikan *URL* melalui akun-akun palsu. Proses

pengambilan jumlah pengirim juga dilakukan normalisasi. Implementasi *psuedocode* proses ini ditunjukan pada Gambar Gambar 4.10. Kode sumber lengkap dapat dilihat pada Lampiran J.

```
1   For mengambil data entrypoint
2       SELECT akun WHERE current entrypoint
3       While baca query akun
4          Memasukan akun ke dalam listAkun
5       Normalisasi: listAkun.count / freq_EP
6       Foreach akun in listAkun
7          UPDATE usertraining SET akun=normalisasi
```

Gambar 4.10 *Pseudocode* proses pencatatan jumlah pengirim *entrypoint URL*

4.2.2.7. Implementasi Fitur Pengambilan Nilai Simpangan Baku *Followers* Pengirim

```
1   For mengambil data entrypoint
2      SELECT akun WHERE current entrypoint
3      WHILE baca query akun
4         Memasukan akun ke dalam listAkun
5      Foreach akun in listAkun
6         SELECT follower WHERE akun
7         Masukkan follower ke list
8      Standar deviasi dari list
9      Normalisasi: sdfollower / 200*sqrt(list.count)
10     Foreach akun in listAkun
11        UPDATE usertraining SET sd_foll = normalisasi
```

Gambar 4.11 *Pseudocode* proses pencatatan nilai simpangan baku *follower* pengirim

Pada Proses ini dilakukan pengambilan nilai simpangan baku *follower* sesama pengirim yang telah mengirimkan *entrypoint* yang sama. Apabila hanya terdapat satu *entrypoint* yang muncul pada *n-tweets* maka otomatis nilai yang dihasilkan adalah 0. Pada proses ini juga dilakukan normalisasi yang telah dijelaskan pada bab III. Implementasi *psuedocode* proses ini ditunjukan pada Gambar 4.11. Kode sumber lengkap dapat dilihat pada Lampiran K.

4.2.2.8. Implementasi Fitur Pengambilan Nilai Simpangan Baku *Friends* Pengirim

Sama halnya dengan pengambilan nilai simpangan baku *follower* pengirim, proses ini juga melakukan pengambilan nilai simpangan baku *friends* pada pengirim. Implementasi *psuedocode* proses ini ditunjukan pada Gambar 4.12. Kode sumber lengkap dapat dilihat pada Lampiran L.

```
1   For mengambil data entrypoint
2      SELECT akun WHERE current entrypoint
3      WHILE baca query akun
4         Memasukan akun ke dalam listAkun
5      Foreach akun in listAkun
6         SELECT friend WHERE akun
7         Masukkan friend ke list
8      Standar deviasi dari list
9      Normalisasi: sdfriend / 200*sqrt(list.count)
10     Foreach akun in listAkun
11        UPDATE usertraining SET sd_fren = normalisasi
```

Gambar 4.12 *Pseudocode* proses pencatatan nilai simpangan baku pada *friend* pengirim

4.2.2.9. Implementasi Fitur Pengambilan Nilai Simpangan Baku Pada Rasio Dari *Followers-Friends* Pengirim

```
1   For mengambil data entrypoint
2      SELECT akun WHERE current entrypoint
3      WHILE baca query akun
4         Memasukan akun ke dalam listAkun
5      Foreach akun in listAkun
6         SELECT friend, follower WHERE akun
8         Jika follower < friend
9             Rasio follower / fren
10        Jika friend < follower
11            Rasio fren / follower
12            Masukkan rasio ke listRasio
13     Standar deviasi dari listRasio
14     Normalize: sdfriend / 200*sqrt(listRasio.count)
15     Foreach akun in listAkun
16        UPDATE usertraining SET sd_rasio = normalize
```

Gambar 4.13 *Pseudocode* proses pencatatan nilai simpangan baku pada rasio dari *followers-friends* pengirim

Untuk implementasi pada proses ini ditunjukan pada Gambar 4.13. Proses ini dilakukan pengambilan nilai simpangan baku pada rasio dari *followers* dan *friends* pengirim. Seperti yang dijelaskan pada bab III, telah dijelaskan perhitungan rasio dari *followers* dan *friends* pengirim. Kode sumber lengkap dapat dilihat pada Lampiran M.

4.2.2.10. Implementasi Fitur Pengambilan Nilai Kemiripan Teks *Tweet* Pada Pengirim

Pada proses ini juga dilakukan pengecekan kemiripan teks pada *tweet* oleh pengirim yang telah mengirimkan *entrypoint* masing-masing. Untuk melakukan perhitungan kemiripan teks, digunakan *Jaccard index*, dimana untuk mengukur kemiripan teks antara kalimat satu dengan kalimat yang lain yang dicek berdasarkan per kata. Implementasi pada proses ini ditunjukan pada Gambar 4.14. Kode sumber lengkap dapat dilihat pada Lampiran N.

```
1   For mengambil data entrypoint
2      SELECT akun WHERE current entrypoint
3      WHILE baca query akun
4         Memasukan akun ke dalam listAkun
5      Foreach akun in listAkun
6         SELECT tweet WHERE akun
7         Words = Split teks berdasarkan spasi
8         Foreach kata in words
9            Memasukkan kedalam listdoc1
10        GET tweet pada timeline pengirim
11        Foreach tweet in tweet.url
12           Words2 = split teks berdasarkan spasi
13           Foreach kata2 in words2
14               Memasukkan kedalam listdoc2
15           Jaccard(listdoc1, listdoc2)
16           Memasukkan jaccard ke listtweet
17           Memasukkan kuadrat dari listdoc2.count
18        Normalisasi:  sum  dari  jaccard  /  akar
    listdoc2.count
19     UPDATE usertraining SET Tsimilar = normalisasi
```

Gambar 4.14 *Pseudocode* proses pengambilan nilai kemiripan teks *tweet* pada pengirim

4.2.3. Implementasi Proses Klasifikasi Fitur Menggunakan Algoritma *Decision Tree*

Pada proses ini merupakan proses komputasi algoritma *Decision Tree* pada fitur yang telah dibentuk. Proses komputasi ini dilakukan untuk membentuk suatu model sebuah pohon yang merepresentasikan kondisi antara *URL* yang aman, dan *URL* yang berbahaya. Dengan mengambil data yang sebelumnya terbentuk yang disebut sebagai data *trining* memodelkan sebuah tree yang akurat untuk mendeteksi status *URL* yang didapat. Hal ini menunjukan pengambilan keputusan akhir didasarkan pada pemodelan tree yang dibentuk. Implementasi pada proses ini dapat dilihat pada Gambar 4.15. Kode sumber lengkap dapat dilihat pada Lampiran O.

```
1  SELECT query urltweet, fitur FROM usertraining
2  WHILE hasil query
3     Mengimplementasikan algoritma decision tree
4     Memasukkan urltweet ke dalam listURL
5     Memasukan label ke dalam listLabel
6  FOR listURL, listLabel
7     UPDATE usertraining SET label WHERE url
```

Gambar 4.15 *Pseudocode* proses klasifikasi fitur menggunakan algoritma *decision tree*

4.2.4. Implementasi Pengiriman Notifikasi

Pada proses ini dilakukan untuk mengirimkan notifikasi setelah melakukan pelabelan status *URL* pada sekumpulan *tweets*. Notifikasi tersebut dikirimkan melalui dua bagian yakni melalui peramban Mozilla firefox dengan menggunakan Greasemonkey *add-ons*, dan juga melalui *email*.

4.2.4.1. Implementasi Peringatan Pada Peramban Mozilla Firefox Menggunakan Greasemonkey *add-ons*

Seperti implementasi yang ditunjukan pada Gambar 4.16, proses ini merupakan proses untuk menunjukan adanya *URL* yang berbahaya yang terkandung di dalam *tweet*. Setelah mendapatkan pelabelan daftar *URL* yang berbahaya, sistem membuat *file*

berformat *javascript* yang isinya memanipulasi halaman *https://twitter.com* pada peramban Mozilla dengan menambahkan *user.js* pada Greasemonkey *add-ons*. Kode sumber lengkap dapat dilihat pada Lampiran P.

```
1   SELECT account, tweet, urltweet FROM username dan
    user training
2       Memasukkan account ke listAkun
3       Memasukkan tweet ke listTweet
4       Memasukkan urltweet ke listUrl
5   Membuat script.js
6      FOR listURL
7         Membuat peringatan
```

Gambar 4.16 *Pseudocode* implementasi peringatan pada peramban Mozilla Firefox menggunakan Greasemonkey *add-ons*

4.2.4.2. Implementasi Notifikasi Melalui *E-mail*

Pada proses ini merupakan proses untuk mengirimkan notifikasi berupa *email* daftar *URL* yang berbahaya. Implementasi pada proses ini dapat dilihat pada Gambar 4.17. Kode sumber lengkap dapat dilihat pada Lampiran Q.

```
1   SELECT account, tweet, urltweet FROM username dan
    user training
2       Memasukkan account ke listAkun
3       Memasukkan tweet ke listTweet
4       Memasukkan urltweet ke listUrl
5   Memasukan formatEmail kedalam teks.txt
6      Menulis subject email
7      FOR listURL
8         Menjelaskan informasi URL berbahaya
9      Menggunakan smtpClient
10        Kirim email ke pengguna
```

Gambar 4.17 *Pseudocode* proses pengiriman notifikasi melalui *email*

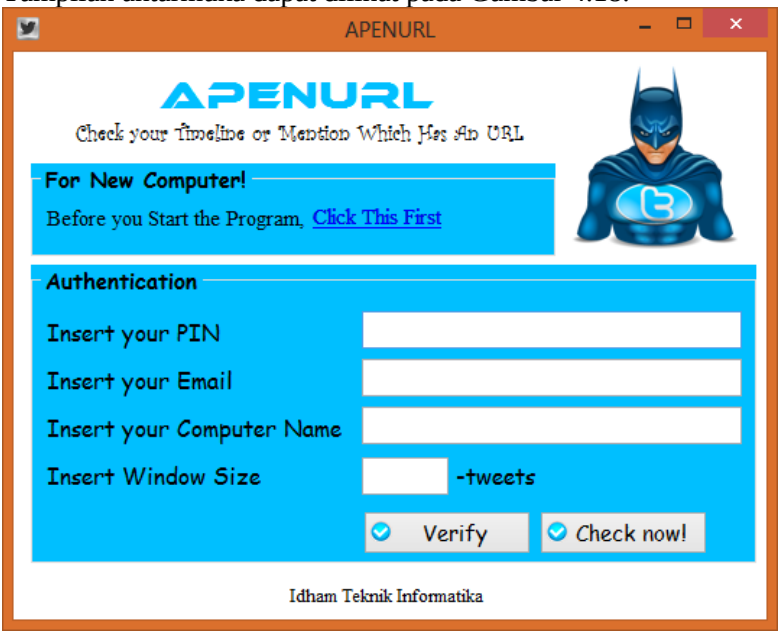
4.3. Implementasi Antarmuka Perangkat Lunak

Implementasi antarmuka perangkat lunak diimplementasikan pada aplikasi *desktop* dimana pengguna melakukan interaksi dengan sistem sesuai dengan rancangan yang telah dibahas pada

BAB III. Penjelasan implementasi antarmuka dijelaskan pada subbab berikut.

4.3.1. Tampilan Antarmuka *Desktop*

Pada tampilan ini, pengguna diminta untuk memasukkan kode otorisasi yang didapatkan pengguna setelah pengguna melakukan *login* pada akun Twitter. Kode otorisasi dimasukkan pada *textbox* yang sudah disediakan, agar dapat masuk ke dalam sistem. Selain itu pengguna juga diminta untuk memasukkan *email* untuk pengiriman notifikasi, nama komputer untuk pengecekkan pada peramban Mozilla, dan *window size* untuk mengecek berapa jumlah *tweets* yang akan dicek. Tombol *verify* yang disediakan akan melakukan proses validasi ketika ditekan dengan mengambil nilai kode otorisasi yang dimasukkan oleh pengguna. Sedangkan tombol *check now* disediakan untuk melakukan pengecekkan. Tampilan antarmuka dapat dilihat pada Gambar 4.18.



Gambar 4.18 Implementasi antarmuka *desktop*

BAB V
UJI COBA DAN EVALUASI

Pada bab ini akan dibahas mengenai uji coba dari segi fungsionalitas dan performa dari aplikasi. Uji coba fungsionalitas dan performa akan dibagi ke dalam beberapa skenario uji coba.

5.1. Lingkungan Uji Coba

Dalam proses uji coba, digunakan beberapa perangkat keras dan perangkat lunak sebagai penunjang kebutuhan yang akan dijelaskan pada subbab berikut.

5.1.1. Perangkat Keras

Kebutuhan perangkat keras yang digunakan dalam uji coba yaitu:

- Prosesor Intel i5-2430M 2.40 GHZ
- Memori 4GB RAM

5.1.2. Perangkat Lunak

Kebutuhan perangkat lunak yang digunakan dalam uji coba yaitu:

- Sistem operasi Windows 8 Ultimate 32-bit pada *server*
- MySQL Server 5.5.27 sebagai basis data
- MySQL Connector Net 6.8.3 sebagai penghubung basis data dengan aplikasi pengembang Microsoft Visual Studio
- XAMPP sebagai *server* yang berjalan pada *localhost* dan digunakan untuk menjalankan basis data MySQL
- Weka 3.6 digunakan untuk mengukur tingkat akurasi dari proses uji coba

5.2. Uji Coba Fungsionalitas

Uji coba fungsionalitas merupakan sebuah pengujian terhadap jalannya fungsi-fungsi utama yang ada pada aplikasi. Uji coba fungsionalitas meliputi semua fitur yang telah dijelaskan pada BAB III, yaitu:

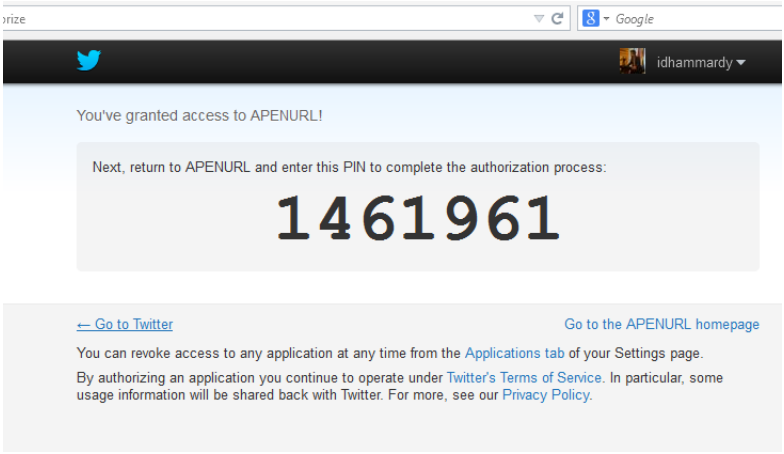
1. *Login* dan mendapatkan kode otorisasi Twitter
2. Proses Skrip Greasemonkey *add-ons* Pada Peramban Mozilla Firefox
3. Proses pengambilan *tweet*
4. Melihat notifikasi pada peramban Mozilla Firefox
5. Melihat notifikasi *URL* berbahaya melalui *email*

5.2.1. *Login* dan Mendapatkan Kode Otorisasi Twitter

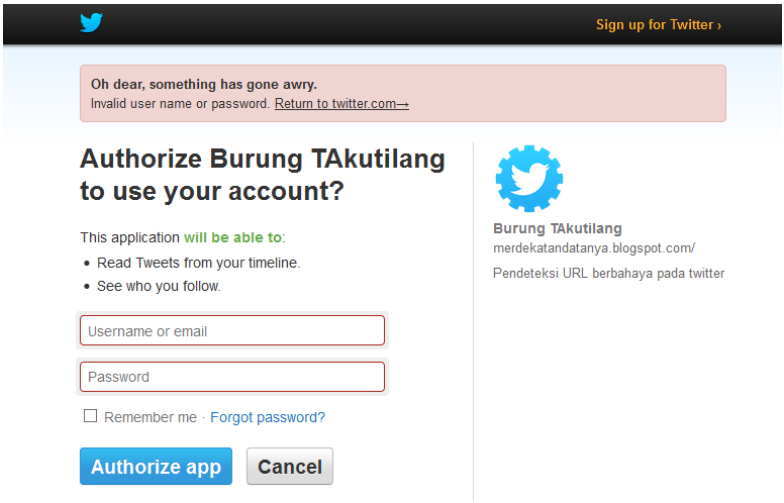
Proses pertama yang dilakukan ketika menjalankan aplikasi *client* adalah proses *login* halaman Twitter pada *default browser* pengguna untuk mendapatkan kode otorisasi. Tabel 5.1 menunjukkan prosedur uji coba yang dilakukan pada proses *login*.

Tabel 5.1 Uji Coba Proses *Login*

ID	**UJ-01**
Nama	Uji Coba Proses *Login* Twitter
Tujuan Uji Coba	Menguji proses pengiriman *login* ke Twitter dan proses mendapatkan kode otorisasi
Kondisi Awal	Aplikasi berjalan
Skenario 1	**Pengguna memasukkan *username* dan *password* akun Twitter yang sesuai dengan benar**
Masukan	*Username* dan *password* yang sesuai
Keluaran yang diharapkan	Kode otorisasi dari Twitter untuk dimasukkan ke dalam sistem oleh pengguna
Hasil Uji Coba	BERHASIL
Skenario 2	**Pengguna memasukkan *username* dan *password* yang tidak sesuai**
Masukan	*Username* dan *password* yang tidak sesuai
Keluaran yang diharapkan	Halaman Twitter yang menunjukkan kesalahan memasukkan *username* dan *password*
Hasil Uji Coba	BERHASIL

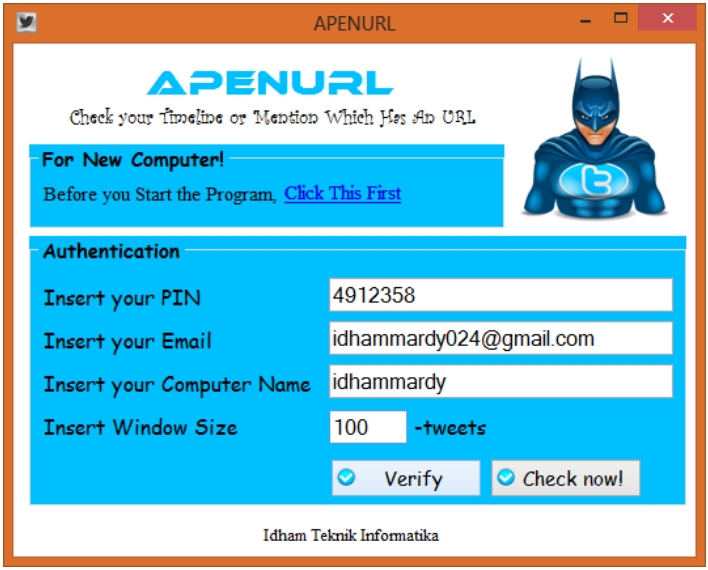


Gambar 5.1 Tampilan login Twitter jika berhasil login

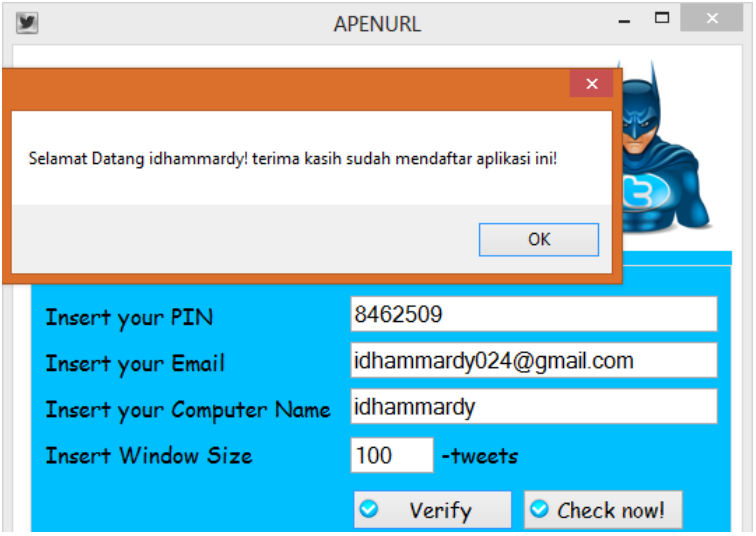


Gambar 5.2 Tampilan pesan kesalahan login

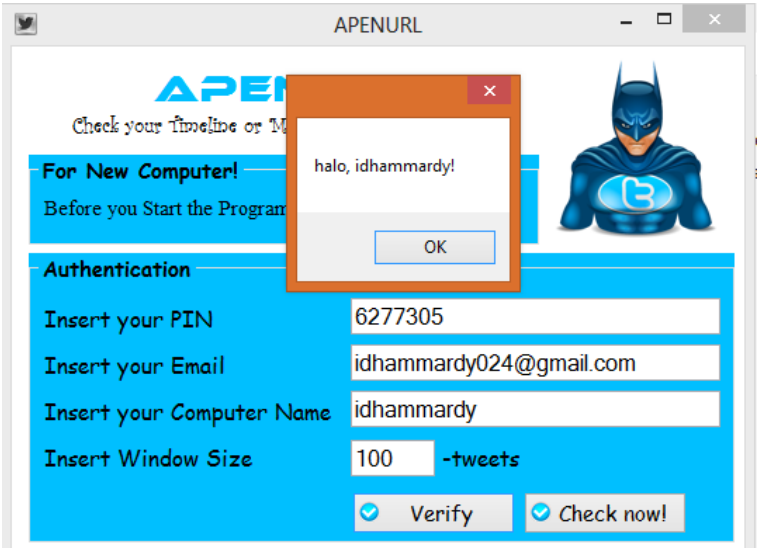
Proses ini bertujuan untuk memberikan ijin aplikasi mengakses *timeline*, dan *mention* pengguna pada *server* Twitter. Proses pengiriman kode otorisasi oleh *server* Twitter tersebut disebut dengan OAuth. Pada saat pengguna memasukkan *username* dan *password* Twitter dengan benar, maka akan ditampilkan menu seperti yang tampak pada Gambar 5.1. Jika proses *login* gagal, maka aplikasi akan menampilkan pesan kesalahan yang tampak pada Gambar 5.2. Setelah mendapatkan kode otorisasi, pengguna dapat melakukan pengisian pada *textbox* pada aplikasi yang telah disediakan. Pada Gambar 5.3 merupakan contoh pengisian pengguna terhadap sistem yang nantinya akan dihubungkan oleh *server* Twitter. Apabila pengguna merupakan pengguna baru pada aplikasi tersebut, maka akan menampilkan *message box* seperti yang ditunjukan pada Gambar 5.4. Sedangkan apabila pengguna merupakan pengguna yang pernah menggunakan aplikasi, maka akan menampilkan *message box* seperti yang ditunjukan pada Gambar 5.5.



Gambar 5.3 Tampilan penulisan pada *textbox* aplikasi



Gambar 5.4 Tampilan pesan pengguna baru



Gambar 5.5 Tampilan pesan pengguna lama

5.2.2. Proses Instalasi Skrip Greasemonkey *add-ons* Pada Peramban Mozilla Firefox

Proses ini dikhususkan bagi siapa saja yang menjalankan aplikasi ini pada komputer yang baru. Skrip tersebut dapat diinstall dengan menekan *link* pada aplikasi. Karena setelah dilakukannya pengecekkan, sistem melakukan *generate javascript* pada user.script.js yang telah terinstall tersebut. Sehingga aplikasi akan mengecek nama *user* pada komputer untuk merubah isi skrip pada user.script.js yang telah diinstall. Tabel 5.2 menunjukan prosedur uji coba yang dilakukan pada proses instalasi *user* skrip pada peramban Mozilla Firefox pada *timeline*. Gambar 5.6 merupakan halaman untuk menginstal skrip Greasemonkey pada peramban Mozilla Firefox. Gambar 5.7 menunjukan proses instalasi, dan Gambar 5.8 menunjukan keberhasilan dari proses instalasi tersebut.

Tabel 5.2 Uji coba instalasi skrip Greasemonkey

ID	**UJ-02**
Nama	Uji coba instalasi *user* skrip Greasemonkey *add-ons*
Tujuan Uji Coba	Menguji keberhasilan instalasi *skrip* Greasemonkey
Kondisi Awal	Aplikasi berjalan, pengguna sudah login
Skenario	**Pengguna menginstal skrip pada link yang disediakan**
Masukan	user.script.js
Keluaran yang diharapkan	Menampilkan notifikasi Greasemonkey bahwa skrip berhasil diinstal
Hasil Uji Coba	BERHASIL

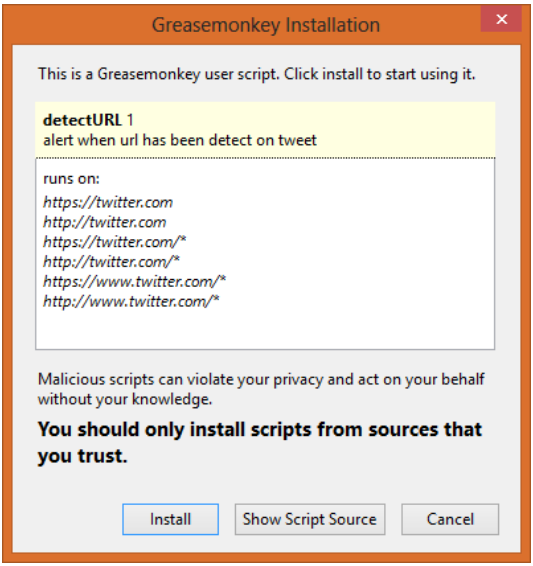
The First you should do before you run APENURL application

first, Install Greasemonkey Add-ons on your Mozilla Firefox Browser

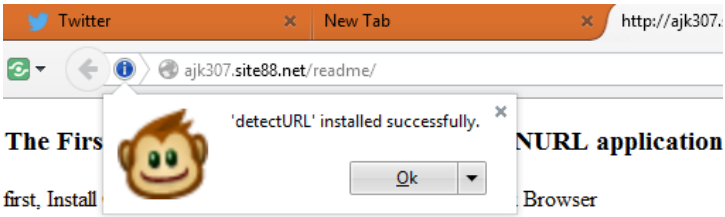
Second, Install my script by **CLICK** detectURL.user.js

Reminder, you only can use the Mozilla Firefox Browser and go to twitter after you start the

Gambar 5.6 Halaman instalasi skrip Greasemonkey



Gambar 5.7 Instalasi Greasemonkey

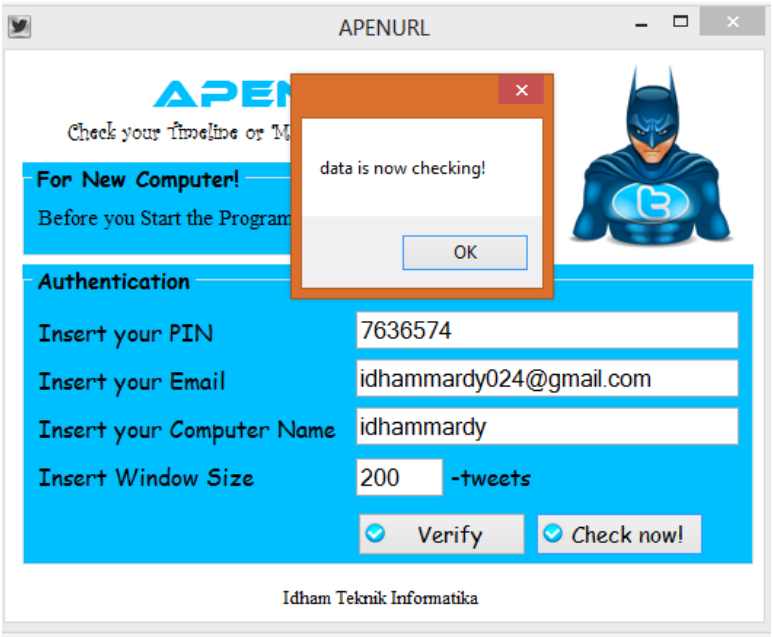


Gambar 5.8 Instalasi Greasemonkey berhasil

5.2.3. Proses Pengambilan *Tweet*

Proses pengambilan *tweet* bertujuan untuk mengambil data *tweet* pada pengguna yang telah melakukan *login*. Pengambilan *tweet* dilakukan untuk mengambil *timeline* dan *mention* dari pengguna yang *tweet*-nya mengandung *URL*. Karena *tweet* yang diambil hanya *tweet* yang mengandung *URL* itulah sehingga dalam pengambilan *tweet* memerlukan waktu yang cukup lama.

Jika proses pengambilan *tweet* telah terkumpul sejumlah *n-tweet*, maka sistem akan menampilkan pesan data sudah siap dilakukan hal ini ditunjukan pada Gambar 5.9. Proses pengambilan *tweet* tersebut dibagi menjadi dua bagian, yakni pengambilan *tweet* pada *timeline*, dan pengambilan *tweet* pada *mention*. Hal ini akan dijelaskan pada sub bab selanjutnya.



Gambar 5.9 Data memulai pengecekan

5.2.2.1 Proses Pengambilan *Tweet* Pada *Timeline*

Pada uji coba ini akan dilakukan proses pengambilan *tweet* pada *timeline* pengguna. Tabel 5.3 menunjukan prosedur uji coba yang dilakukan pada proses pengambilan *tweet* pada *timeline*. Gambar 5.10 menunjukan data *tweet* yang diambil dari *mention*, yang telah disimpan kedalam basis data.

Tabel 5.3 Uji coba pengambilan *tweet* pada *timeline*

ID	**UJ-03**
Nama	Uji coba pengambilan *tweet* pada *timeline*
Tujuan Uji Coba	Menguji pengambilan data *tweet* pada *timeline*
Kondisi Awal	Aplikasi berjalan, pengguna sudah login
Skenario	**Sistem mengunduh *tweet* pada *timeline* pengguna**
Masukan	Data *tweet* yang mengandung *URL*
Keluaran yang diharapkan	Sistem menampilkan pesan bahwa data siap dilakukan pengecekan setelah terkumpul sejumlah *n-tweet*
Hasil Uji Coba	BERHASIL

crawl_type	account	tweet	urltweet
TimeLine	HinduismUpdate	President Mukherjee to receive a copy of encyclope...	http://t.co/nExCGMtyk2
TimeLine	detikcom	Italia Kalah, Inggris Tersingkir http://t.co/FThYP...	http://t.co/FThYPKheJY
TimeLine	detikcom	Prancis Gilas Swiss 5-2 http://t.co/7gaTGlnH33	http://t.co/7gaTGlnH33
TimeLine	Yudek_archie88	RT @adecsaraswati: AIGUUUUUUU "kairamiisu: EEEPS B...	http://t.co/GoA4kKBkjo
TimeLine	detikcom	Seknas Jokowi Desak Kapolri Tuntaskan Kasus Transk...	http://t.co/OiU5o1kmQ7
TimeLine	kompascom	Fadli Zon: Wiranto Jenderal yang Pengecut dan Pecu...	http://t.co/K2soqTAE42
TimeLine	Richard_Florida	RT @markbyrnes525: Donald Trump's op-ed defending ...	http://t.co/k2rsJmFUon
TimeLine	kompascom	Perancis Pesta Gol ke Gawang Swiss http://t.co/ilm...	http://t.co/ilmbL30ilb
TimeLine	detikcom	Potongan Kepala Tanpa Mata, Hidung dan Telinga Ber...	http://t.co/aiSOX95jHJ
TimeLine	FastCoCreate	This is how they're watching the #WorldCup in Berl...	http://t.co/Zf11EkZdhv
TimeLine	FastCoDesign	Look at Garamond in a different light in these gor...	http://t.co/1NBg3t79cr

Gambar 5.10 Data tweet pada timeline

5.2.2.2. Proses Pengambilan *Tweet* Pada *Mention*

Selain dilakukan uji coba pengambilan *tweet* pada *timeline*, juga dilakukan proses pengambilan *tweet* pada *mention* pengguna. Tabel 5.4 menunjukan prosedur uji coba yang dilakukan pada proses pengambilan *tweet* pada *mention*. Gambar 5.11 menunjukan data *tweet* yang diambil dari *mention*, yang telah disimpan kedalam basis data.

Tabel 5.4 Uji coba pengambilan *tweet* pada *mention*

ID	**UJ-04**
Nama	Uji coba pengambilan *tweet*
Tujuan Uji Coba	Menguji pengambilan data *tweet* pada *mention*
Kondisi Awal	Aplikasi berjalan, pengguna sudah login
Skenario	**Sistem mengunduh *tweet* pada *mention* pengguna**
Masukan	Data *tweet* yang mengandung *URL*
Keluaran yang diharapkan	Sistem menampilkan pesan bahwa data siap dilakukan pengecekan setelah terkumpul sejumlah *n-tweet*
Hasil Uji Coba	BERHASIL

crawl_type	account	tweet	urltweet
Mention	marroland110	@sdarmaputra jangan dihiraukan, untuk simulasi uji...	http://t.co/kTu8t2AouF
Mention	follicur110	@sdarmaputra maaf ya teman-teman, sedang uji perco...	http://t.co/kNIYKiUGXs
Mention	rey_ryan	Meja pimpong meja rapat "@sdarmaputra: Meeting (wi...	https://t.co/z3IjFWPyuJ
Mention	IGst_Angga	Beginilah hari2ku "@sdarmaputra: Meeting (with @re...	https://t.co/ZdfaWByqX
Mention	RioPurboyo	Workshop u' Anda yg ingin mahir gunakan #persuasi ...	https://t.co/DIG6tzsHR
Mention	dalaaaaaaa	Kelas pengganti RO ._. (with @sdarmaputra, @adhipo...	https://t.co/wDYrvA1Sg
Mention	adhipoer	RT @sdarmaputra: H-1 bung! semangat! -- SELAYANG P...	http://t.co/cgSXvYidL2

Gambar 5.11 Data tweet pada mention

5.2.4. Proses Ekstraksi Korelasi Rantai *Redirect URL*

Pada uji coba ini akan dilakukan proses pencarian *entrypoint URL*. Ketika data sistem telah mengumpulkan data *tweet* sejumlah *n-tweets*, dan memulai untuk pengecekan, data kemudian diambil dan dicocokin antara *tweet* yang mengandung *URL* satu dengan yang lain. Seperti yang dijelaskan pada bab sebelumnya, Apabila terdapat lebih dari satu *URL* pada rantai *redirect URL* satu dengan *URL* yang lain, maka bisa dikatakan *URL* tersebut adalah *entrypoint*. Tabel 5.5 menunjukkan prosedur uji coba yang dilakukan pada proses melakukan pencarian *entrypoint URL*.

Tabel 5.5 Uji coba proses ekstraksi korelasi rantai *redirect URL*

ID	**UJ-05**
Nama	Uji coba proses ekstraksi korelasi rantai *redirect URL* dengan pencarian *entrypoint URL*
Tujuan Uji Coba	Menguji proses pencarian *entrypoint URL* pada sekumpulan *n-tweets*
Kondisi Awal	Aplikasi berjalan, pengambilan *tweet* selesai, pengecekan dimulai
Skenario	**Sistem melakukan pencarian *entrypoint URL***
Masukan	Kumpulan rantai *redirect URL* pada *n-tweets*
Keluaran yang diharapkan	List *entrypoint URL* yang
Hasil Uji Coba	BERHASIL

Selanjutnya dalam melakukan pengujian pada sistem, dilakukan pengecekan melalui *output* yang dihasilkan pada Visual Studio dengan menggunakan *console.writeline*. sehingga dapat dipantau korelasi antar rantai *redirect URL* yang menunjukan adanya *entrypoint* ditiap-tiap *tweet* yang mengandung rantai *redirect URL* tersebut. Pencarian *entrypoint URL* dipengaruhi jumlah *n-tweets* yang dicek. Semakin banyak *URL* yang akan dicek, waktu yang diperlukan juga cukup lama. Untuk keluaran uji coba proses pencarian *entrypoint URL* yang dihasilkan dapat ditunjukan pada Gambar 5.12.

Gambar 5.12 *entrypoint* dan frekuensi *entrypoint URL*

5.2.5. Proses Ekstraksi Konteks Informasi *Tweet* Pada Pengirim

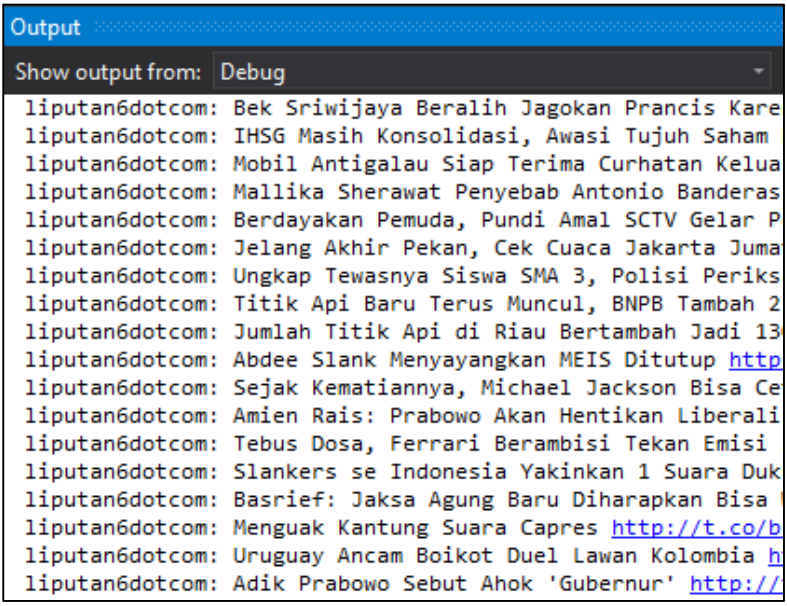
Pada uji ini akan dilakukan proses pengambilan konteks informasi *tweet* pada pengirim. Proses ini diambil untuk menentukan mendapatkan nilai dari fitur yang dihasilkan, seperti mendapatkan nilai standar deviasi antara *friend* dan *follower*, dan juga kemiripan *tweet* pada isi *timeline* pengirim. Untuk output yang dihasilkan untuk mendapatkan *friends* dan *follower* pada pengirim ditunjukan pada Gambar 5.13. sedangkan untuk menlihat output yang dihasilkan untuk mendapatkan teks *tweet* pada isi timeline pengirim ditunjukan pada Gambar 5.14. Tabel 5.6 menunjukan prosedur uji coba yang dilakukan pada proses ekstraksi konteks informasi *tweet* pada pengirim.

Tabel 5.6 Uji coba proses ekstraksi konteks informasi *tweet*

ID	**UJ-06**
Nama	Uji Coba Proses Ekstraksi Konteks Informasi *Tweet* Pada Pengirim
Tujuan Uji Coba	Menguji proses pengambilan data informasi *tweet* pada pengirim
Kondisi Awal	Aplikasi berjalan, pengambilan data *tweet* berjalan
Skenario 1	**Pengambilan *friends-followers* pengirim**
Masukan	Akun Twitter pengirim
Keluaran yang diharapkan	Data ditampung kedalam *database*
Hasil Uji Coba	BERHASIL
Skenario 2	**Pengambilan *tweet* pada *timeline* pengirim**
Masukan	Akun Twitter pengirim
Keluaran yang diharapkan	Data ditampung kedalam list, dan dicocokkan kemiripan teks
Hasil Uji Coba	BERHASIL

account	following	follower
VIVAbola	297	597623
BeasiswaGratis	33	261301
liputan6dotcom	25	1111496
MobilePunch	2571	400145
malaysiakini	15	217985
eventsurabaya	59072	87460
detikTravel	44	230394
detikcom	35	7363616
dokterMade	813	51749
itokwinursito	617	322
VIVAnews	35	1847761

Gambar 5.13 Data *followers* dan *friends* pada pengirim



Gambar 5.14 Data isi *tweet* pada pengguna

5.2.6. Proses Ekstraksi dan Normalisasi Fitur

Pada uji coba ini dilakukan proses ekstraksi dan normalisasi fitur. Terdapat 10 fitur yang diambil setelah dapat melakukan proses ekstraksi rantai *redirect URL* dan ekstraksi konteks informasi *tweet* pada pengirim. Setiap nilai yang diambil dari masing-masing proses akan dilakukan normalisasi. Normalisasi merupakan proses yang digunakan untuk menentukan klasifikasi dari fitur yang didapat guna meminimalisir adanya redundansi pada data. Dalam hal ini fitur-fitur akan dinormalisasikan antara nol hingga satu. Proses yang akan diuji antara lain:

1. Fitur pengambilan nilai panjang *redirect URL*.
2. Fitur pengambilan nilai frekuensi *entrypoint URL*.
3. Fitur pengambilan nilai letak *entrypoint URL*.
4. Fitur pengambilan nilai perbedaan inisial *URL* pada *entrypoint URL* yang sama.
5. Fitur pengambilan nilai perbedaan *landing URL* pada *entrypoint URL* yang sama.
6. Fitur pengambilan jumlah pengirim *entrypoint URL*.
7. Fitur pengambilan nilai simpangan baku *follower* pengirim.
8. Fitur pengambilan nilai simpangan baku *friend* pengirim.
9. Fitur pengambilan nilai simpangan baku pada rasio dari *friend-follower* pengirim.
10. Fitur pengambilan nilai kemiripan teks *tweet* pada pengirim.

5.2.6.1. Fitur Pengambilan Nilai Panjang *Redirect URL*

Seperti yang dijelaskan pada BAB III, bahwa untuk mengetahui kebiasaan dan membedakan antara *URL* berbahaya dan *URL* yang aman dimulai dari panjang *redirect URL* tersebut, karena *URL* yang mencurigakan adalah *URL* tersebut mempunyai rantai *redirect* yang lebih panjang dari *URL* biasanya. Berdasarkan analisa sebelum-sebelumnya, sebagaian besar didapat kurang dari tujuh pengalihan *URL* dalam satu rantai *redirect URL*. Sehingga batas atas nilai dari panjang *redirect* yakni 7. Karena berdasarkan

analisa sebelum-sebelumnya, sebagaian besar didapat kurang dari tujuh pengalihan *URL* dalam satu rantai *redirect URL*, sehingga untuk nilai normalisasinya yakni $\boldsymbol{mi}\,\mathbf{n}(\boldsymbol{l},\mathbf{7})\,/\mathbf{7}$. Tabel 5.7 menunjukkan bentuk normalisasi pada proses melakukan pengambilan nilai panjang *redirect URL*.

Tabel 5.7 normalisasi nilai panjang *redirect URL*

Panjang *Redirect*	Perhitungan normalisasi	nilai
2	2/7	0.28
3	3/7	0.43
4	4/7	0.57
5	5/7	0.71
6	6/7	0.85
7	7/7	1

Gambar 5.15 merupakan hasil uji coba contoh beberapa data yang didapat setelah melakukan fitur pengambilan nilai panjang *redirect URL* yang terjadi. Data tersebut dilakukan *query* pada basis data pada data yang sudah dilakukan pengecekkan.

detect_on	urltweet	length_url
TimeLine	http://t.co /u0JkTrVfzx	0.43
TimeLine	http://t.co /F2S3YRKPLN	1
TimeLine	http://t.co /sicG1s7aRT	0.57

Gambar 5.15 Nilai normalisasi panjang *redirect URL*

Bila dibuktikan pada panjang *redirection URL*, ialah seperti yang ditunjukan pada Gambar 5.16 dimana menunjukan inisial *URL http://t.co/u0JkTrVfzx*, Gambar 5.17 dimana menunjukan

inisial *URL http://t.co/F2S3YRKPLN*, dan Gambar 5.18 dimana menunjukan inisial *URL http://t.co/sicG1s7aRT*.

urltweet	urlredirect	posisi	length
http://t.co/u0JkTrVfzx	http://t.co/u0JkTrVfzx	1	3
http://t.co/u0JkTrVfzx	http://bit.ly/1qzjf2l	2	3

Gambar 5.16 Panjang *redirect URL http://t.co/u0JkTrVfzx*

urltweet	urlredirect	posisi	length
http://t.co/F2S3YRKPLN	http://t.co/F2S3YRKPLN	1	7
http://t.co/F2S3YRKPLN	http://bit.ly/1hQTvM8	2	7
http://t.co/F2S3YRKPLN	http://twilinks.net/rd/index.php?url=http://c.mbaf...	3	7
http://t.co/F2S3YRKPLN	http://c.mbaf.jp/r/1vq6ajk/	4	7
http://t.co/F2S3YRKPLN	http://adinsight.jp/662205001?misc=1120753922af32e...	5	7
http://t.co/F2S3YRKPLN	http://app-adforce.jp/ad/p/r?_site=11426&_article=...	6	7

Gambar 5.17 Panjang *redirect URL http://t.co/F2S3YRKPLN*

urltweet	urlredirect	posisi	length
http://t.co/sicG1s7aRT	http://t.co/sicG1s7aRT	1	4
http://t.co/sicG1s7aRT	http://bit.ly/Upk9RN	2	4
http://t.co/sicG1s7aRT	http://vk.com/page-45729788_47654625	3	4

Gambar 5.18 Panjang *redirect URL http://t.co/sicG1s7aRT*

5.2.6.2. Fitur Pengambilan Nilai Frekuensi *Entrypoint URL*

Proses pendapatan nilai frekuensi *entrypoint URL* berlangsung ketika mendapatkan *entrypoint*. Sehingga ketika masing-masing *tweet* mempunyai nilai frekuensi *entrypoint*, setelah itu dimasukkan kedalam *list*. Pada *window size* atau jumlah *n-tweets* yang diambil dan dilakukan pada implementasi ini adalah 100 data *tweet*, sehingga untuk normalisasinya ialah $\boldsymbol{n/w}$. Seperti yang ditunjukan pada Gambar 5.12, nilai frekuensi *entrypoint URL* yang didapat dari inisial *URL http://t.co/F2S3YRKPLN* ialah 0.05. hal ini didapat karena normalisasi frekuensi *entrypoint*:

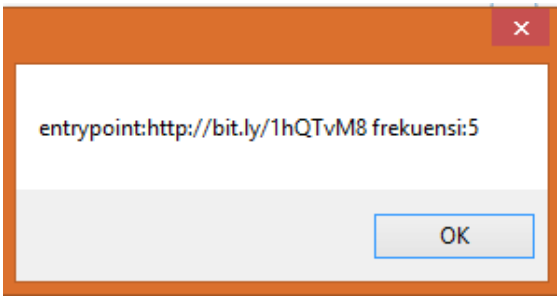
$$\frac{n}{w} = \frac{5}{100} = 0.05$$

Bila dibuktikan *entrypoint URL* yang terkandung didalam rantai *redirect URL* tersebut ditunjukan pada Gambar 5.19 dimana pada inisial *URL http://t.co/F2S3YRKPLN* mempunyai *entrypoint URL* yang mirip dengan inisial *URL* yang lain dengan *entrypoint http://bit.ly/1hQTvM8*.

urltweet	urlredirect
http://t.co/F2S3YRKPLN	http://bit.ly/1hQTvM8
http://t.co/z2FyHyjU0v	http://bit.ly/1hQTvM8
http://t.co/F2S3YRKPLN	http://bit.ly/1hQTvM8
http://t.co/6AMcrLEasS	http://bit.ly/1hQTvM8
http://t.co/F2S3YRKPLN	http://bit.ly/1hQTvM8

Gambar 5.19 Tercatat *entrypoint URL*

Pada Gambar 5.20 dapat menunjukan keberhasilan pencarian *entrypoint* serta mendapatkan jumlah frekuensi yang terjadi dengan menggunkan *messagebox button*.



Gambar 5.20 Frekuensi dan *entrypoint URL* dalam bentuk *messagebox*

5.2.6.3. Fitur Pengambilan Nilai Letak *Entrypoint URL*

Proses pengambilan nilai letak *entrypoint URL* didapat dari ketika *entrypoint* telah ditemukan. Perlu diketahhui bahwa *entrypoint URL* yang mencurigakan tidak terletak pada akhir rantai *redirect URL* karena *URL* tersebut mengalami pengalihan kondisional untuk mengarahkan pengunjung pada *landing URL* yang berbeda. Nilai letak *entrypoint* dinormalisasikan menjadi $\boldsymbol{p/l}$, seperti yang telah dijelaskan pada bab III. Dimana $\boldsymbol{p}$ dinotasikan sebagai letak *entrypoint URL* pada suatu rantai *redirect URL*, dan $\boldsymbol{l}$ dinotasikan sebagai panjang rantai *redirect URL*. Gambar 5.21 merupakan uji coba pengambilan letak *entrypoint URL*.

urltweet	urlredirect	posisi
http://t.co/F2S3YRKPLN	http://t.co/F2S3YRKPLN	1
http://t.co/F2S3YRKPLN	http://bit.ly/1hQTvM8	2
http://t.co/F2S3YRKPLN	http://twilinks.net/rd/index.php?url=http://c.mbaf...	3
http://t.co/F2S3YRKPLN	http://c.mbaf.jp/r/1vq6ajk/	4
http://t.co/F2S3YRKPLN	http://adinsight.jp/662205001?misc=1120753922af32e...	5
http://t.co/F2S3YRKPLN	http://app-adforce.jp/ad/p/r?_site=11426&_article=...	6

Gambar 5.21 Pengecekkan letak entrypoint URL

Jika pada Gambar 5.18 menjelaskan bahwa *entrypoint* terletak pada posisi kedua pada rantai *redirect URL*, maka nilai yang didapat ditunjukan pada Gambar 5.22 dimana normalisasi pada nilai ini ialah:

$$\frac{p}{l} = \frac{2}{7} = 0.2857143$$

urltweet	length_url	frequencyEP	position_EP
http://t.co /F2S3YRKPLN	1	0.05	0.285714285714286

Gambar 5.22 Normalisasi terhadap letak *entrypoint URL* yang terkandung

5.2.6.4. Fitur Pengambilan Nilai Perbedaan Inisial *URL* Pada *Entrypoint URL* yang Sama

Seperti yang dilakukan pada proses sebelumnya, proses pengambilan nilai perbedaan inisial *URL* didapat ketika *entrypoint* juga telah ditemukan. Satu Akun dengan akun yang lain ketika mendistribusikan *tweet*nya yang mengandung *URL*, akan meng-*generate shortening URL* yang dijadikan inisial *URL* yang berbeda–beda meskipun mengarah ke *entrypoint* yang sama, sehingga dengan sistem ini melakukan proses pengecekan dan uji coba menghitung jumlah inisial *URL* yang berbeda ketika mengalami pengalihan ke *entrypoint* yang sama Normalisasi yang dilakukan adalah $\boldsymbol{i/n}$, dimana $\boldsymbol{n}$ merupakan jumlah *entrypoint* yang muncul dan $\boldsymbol{i}$ merupakan jumlah inisial *URL* yang mengandung *entrypoint URL* tersebut. Gambar 5.23 merupakan uji coba pengambilan nilai inisial *URL* yang mempunyai kesamaan *entrypoint URL*.

urltweet	differentInitURL
http://t.co/F2S3YRKPLN	0.6

Gambar 5.23 Pengambilan nilai normalisasi pada inisial *URL*

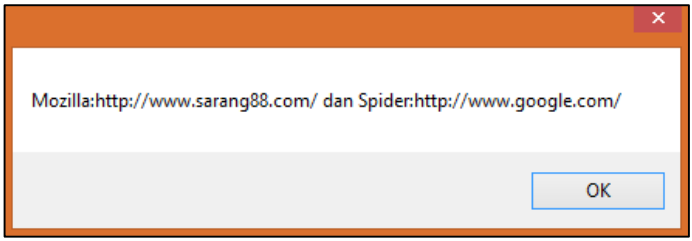
Bila dibuktikan pada catatan pada basis data, inisial *URL http://t.co/ F2S3YRKPLN* mempunyai tiga inisial *URL* yang sama, hal ini ditunjukan pada Gambar 5.24.

urltweet	urlredirect
http://t.co/F2S3YRKPLN	http://bit.ly/1hQTvM8
http://t.co/z2FyHyjU0v	http://bit.ly/1hQTvM8
http://t.co/F2S3YRKPLN	http://bit.ly/1hQTvM8
http://t.co/6AMcrLEasS	http://bit.ly/1hQTvM8
http://t.co/F2S3YRKPLN	http://bit.ly/1hQTvM8

Gambar 5.24 Pengecekkan inisial *URL* pada *entrypoint URL* yang sama

5.2.6.5. Fitur Pengambilan Nilai Perbedaan *Landing URL* Pada *Entrypoint URL* yang Sama

Pada proses ini, sistem melalukan pengecekan jumlah *landing URL* pada setiap *entrypoint URL* yang dihasilkan. Apabila terindikasi bahwa pada suatu tautan mengalami perbedaan *landing page* maka disimpulkan bahwa *landing URL* yang dihasilkan lebih dari satu. Pada pengujian proses pengambilan nilai perbedaan *landing URL*, dilakukan dengan menggunakan *messagebox* pada *library windows form* ketika terjadinya perbedaan *landing URL*. sehingga untuk mengecek perbedaan *landing URL* dilakukan dengan menggunakan *UserAgent* peramban normal dan *UserAgent crawler*. Setelah mendapatkan datanya, dilakukan normalisasi yakni jika *entrypoint* yang muncul sebanyak $\boldsymbol{n}$ melakukan perbedaan pengalihan *landing URL* sebanyak $\boldsymbol{\lambda}$, maka normalisasinya adalah $\boldsymbol{\lambda/n}$. Gambar 5.25 merupakan uji coba pengambilan melalui tampilan *messagebox*.



Gambar 5.25 Pengecekkan perbedaan *landing URL* dalam bentuk *messagebox*

Bila dicek *entrypoint URL* yang dihasilkan sehingga mengarah ke *landing URL* yang berbeda, maka dapat dilakukan analisis terhadap *URL* apa saja yang terkandung di dalam rantai *redirect URL*. Hal ini ditunjukan pada Gambar 5.26 pada rantai *redirect URL* yang terkandung dan *entrypoint URL* yang ditemukan pada Gambar 5.27.

urltweet	urlredirect
http://t.co/kTu8t2AouF	http://t.co/kTu8t2AouF
http://t.co/kTu8t2AouF	http://bit.ly/1oXKXSn
http://t.co/kTu8t2AouF	http://medeniurl.comlu.com/masuk/

Gambar 5.26 Pengecekkan rantai *redirect URL*

urltweet	urlredirect
http://t.co/HcfcqSI1XJ	http://medeniurl.comlu.com/masuk/
http://t.co/kTu8t2AouF	http://medeniurl.comlu.com/masuk/
http://t.co/kNIYKiUGXs	http://medeniurl.comlu.com/masuk/

Gambar 5.27 Pencarian *entrypoint* yang mengarahkan *URL* ke arah *landing URL* yang berbeda

Pada kasus ini, terindikasi bahwa frekuensi *entrypoint URL* yang dihasilkan ialah tiga. Namun mengalihkan kearah dua *landing URL* yang berbeda. sehingga untuk normalisasi hal ini ditunjukan pada Gambar 5.28, dimana:

$$\frac{\lambda}{l} = \frac{2}{3} = 0.66666667$$

urltweet	differentLanding
http://t.co/kTu8t2AouF	0.666666666666667

Gambar 5.28 Nilai normalisasi pada perbedaan *landing URL*

5.2.6.6. Fitur Pengambilan Jumlah Pengirim *Entrypoint URL*

Pada pengujian proses pengambilan jumlah pengirim *entrypoint URL* dapat diketahui ketika data sudah tersimpan pada basis data. Perlu diketahui bahwa banyak penyerang yang membuat akun–akun palsu untuk mendistribusikan *tweet*-nya yang

mengandung *URL*. jumlah pengirim dihitung berdasarkan seberapa banyak frekuensi *entrypoint* yang muncul, sehingga ketika terdapat lebih dari satu nilai frekuensi *entrypoint* akan tetapi diketahui bahwa *entrypoint URL* tersebut berasal dari pengirim yang sama maka hanya diambil salah satunya Nilai tersebut dinormalisasikan menjadi $\boldsymbol{\alpha/n}$ dimana $\boldsymbol{n}$ merupakan jumlah *entrypoint* yang dihasilkan, dan $\boldsymbol{\alpha}$ merupakan banyaknya jumlah pengirim yang telah mengirimkan *entrypoint URL*.

Dalam pengujian pengambilan jumlah pengirim *entrypoint URL*, terdapat dua kasus berbeda, antara jumlah pengirim yang berbeda, maupun yang sama. Berikut Gambar 5.29 merupakan hasil uji coba yang dihasilkan untuk mendapatkan nilai jumlah pengirim *entrypoint URL*.

account	urltweet	urlredirect
aya1213tora	http://t.co/F2S3YRKPLN	http://bit.ly/1hQTvM8
marvel_0908	http://t.co/z2FyHyjU0v	http://bit.ly/1hQTvM8
sa_yaka12	http://t.co/F2S3YRKPLN	http://bit.ly/1hQTvM8
yamachi8107	http://t.co/6AMcrLEasS	http://bit.ly/1hQTvM8
gbxyz1	http://t.co/F2S3YRKPLN	http://bit.ly/1hQTvM8

Gambar 5.29 Pengirim yang telah mengirimkan *entrypoint URL* yang sama

Diketahui pengirim yang telah mengirimkan *URL* dengan *entrypoint URL* yang sama, terdapat lima pengirim, dimana frekuensi *entrypoint URL* juga lima *URL* yang sama. Sehingga untuk normalisasinya $\boldsymbol{\alpha/n}$ yakni 1. Hal ini dibuktikan pada Gambar 5.30.

urltweet	attack_number
http://t.co/F2S3YRKPLN	1

Gambar 5.30 Nilai dari jumlah pengirim *entrypoint* yang telah dilakukan normalisasi

Selanjutnya, pada uji coba kasus yang kedua, dimana pengirim yang telah mengirimkan *entrypoint* yang sama, merupakan pengirim yang sama. Seperti yang ditunjukan pada Gambar 5.31.

account	urltweet	urlredirect
arenanews1	http://t.co/WP7PnXtYT0	http://bit.ly/1m4ZAV3
arenanews1	http://t.co/6SXYESCwG5	http://bit.ly/1m4ZAV3

Gambar 5.31 Pengirim yang telah mengirimkan *entrypoint URL* yang sama

Pada kasus tersebut, telah dihitung normalisasi yang didapat, yakni jumlah pengirim yang telah mengirimkan *entrypoint* $(\boldsymbol{\alpha})$ adalah satu, dan mempunyai frekuensi *entrypoint* $(\boldsymbol{n})$ sebanyak dua, maka normalisasinya yakni 0.5. hal ini ditunjukan pada Gambar 5.32.

urltweet	attack_number
http://t.co/WP7PnXtYT0	0.5

Gambar 5.32 Nilai dari jumlah pengirim *entrypoint* yang telah dilakukan normalisasi

5.2.6.7. Fitur Pengambilan Nilai Simpangan Baku *Followers* Pengirim

Akun-akun palsu yang dibuat oleh penyerang biasanya mempunyai angka kemiripan yang besar. Hal ini mempertimbangkan pada status akun seperti *followers* dan *friends* dari akun tersebut. Pada proses ini, *followers* dari akun tersebut akan dihitung simpangan baku *followers* antara akun satu dengan akun yang lain. Simpangan baku tersebut digunakan untuk mengecek ukuran nilai yang lazim dari *follower* yang tersebar pada akun-akun pengirim untuk mengecek kemiripan angka *followers* akun satu dengan yang lainnya pada *entrypoint URL* yang sama.

Sehingga pada uji coba proses pengambilan nilai simpangan baku pada *follower* pengirim, mengecek jumlah *follower* pada akun pengirim. Pada saat pengambilan data *tweet*, sistem juga otomatis menyimpan daftar *follower* dan *following* dari pengirim. Oleh sebab itu, ketika terdapat kesamaan *entrypoint URL* yang lebih dari satu maka simpangan baku tersebut dapat dihitung, namun apabila hanya terdapat satu *entrypoint URL* maka nilai simpangan baku tidak dapat dihitung atau bernilai nol.
Normalisasi:

$$\boldsymbol{min}\left(\frac{\boldsymbol{std}(\#\boldsymbol{followers})}{\boldsymbol{200}\sqrt{\boldsymbol{n}}}, \boldsymbol{1}\right)$$

Standar deviasi:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

Pada kasus ini mengambil contoh pengirim *entrypoint URL* yang ditunjukan paa Gambar 5.33, dimana mempunyai frekuensi *entrypoint URL* sejumlah lima. Sehingga dari dicari *follower* dari pengrim yang telah mengirimkan *URL* yang terindikasi *entrypoint URL* yang sama tersebut.

account	urltweet	urlredirect
AgusAdiWirawan	http://t.co/YixMZxmA2K	http://bit.ly/asistenPJK04
aldoalase	http://t.co/9L3SF4Q921	http://bit.ly/asistenPJK04
novitanataa	http://t.co/4POzUvuc0W	http://bit.ly/asistenPJK04
ebeckz22	http://t.co/BycuFML6aG	http://bit.ly/asistenPJK04
happyach	http://t.co/D9KJnTwqay	http://bit.ly/asistenPJK04

Gambar 5.33 Pengirim yang telah mengirimkan *entrypoint URL* yang sama

Setelah didapat pengirim yang mengirimkan *entrypoint URL* tersebut, dicari *follower* yang telah tersimpan pada basis data. Hal ini ditunjukan pada Gambar 5.34.

account	follower
AgusAdiWirawan	174
aldoalase	377
novitanataa	243
ebeckz22	392
happyach	539

Gambar 5.34 Jumlah *followers* pada masing-masing pengirim *entrypoint URL*

Dari masing – masing nilai jumlah *followers* yang didapat, memulai perhitungan standar deviasi dengan menghitung rata-rata dari masing-masing jumlah *follower*, dan menghitung varian nilai dari masing-masing nilai tersebut.

Rata-rata: $\frac{174+377+243+392+539}{5} = \frac{1725}{5} = 345$

Varian masing – masing *followers* ditunjukan pada Tabel 5.8.

Tabel 5.8 nilai varian terhadap rata-rata yang diperoleh

followers	varian	hasil
174	$(174-345)^2$	29241
377	$(377-345)^2$	1024
243	$(243-345)^2$	10404
392	$(392-345)^2$	2209
539	$(539-345)^2$	37636
	jumlah	80514

Standar deviasi: $\sqrt{\frac{80514}{5}} = \sqrt{16102.8} = 126.8968$

Normalisasi: $\mathbf{min}(\frac{126.8968}{200\sqrt{5}}, 1) = 0.28374988986782$

account	urltweet	SD_followers
AgusAdiWirawan	http://t.co/YixMZxmA2K	0.28375
aldoalase	http://t.co/9L3SF4Q921	0.28375
novitanataa	http://t.co/4POzUvuc0W	0.28375
ebeckz22	http://t.co/BycuFML6aG	0.28375
happyach	http://t.co/D9KJnTwqay	0.28375

Gambar 5.35 Nilai yang didapat pada simpangan baku *follower* setelah dilakukan normalisasi

Seperti yang ditunjukan pada Gambar 5.35, Setelah melakukan perhitungan standar deviasi, dihitung normalisasi pada persamaan tersebut. perlu diketahui bahwa nilai 200 didapat dari kemungkinan akun-akun palsu yang didapat dalam melakukan *follow* mencapai kurang lebih 200 pengguna.

5.2.6.8. Fitur Pengambilan Nilai Simpangan Baku *Friends* Pengirim

Seperti yang dijelaskan pada sub bab sebelumnya yang dilakukan perhitungan simpangan baku pada *followers* pengirim, juga dilakukan perhitungan pencarian nilai simpangan baku pada *friend* pengirim untuk mengecek nilai dari pesebaran jumlah *friends* dari pengirim. Hal ini akan diketahui bahwa perbedaan akun-akun palsu maupun bukan dapat dilihat dari kemiripan angka pada *friend* pengirim yang tersebar pada Twitter.

Sehingga untuk mengambil nilai simpangan baku pada *friend*, dilakukan normalisasi yakni:

$$\boldsymbol{min}\left(\frac{\boldsymbol{std(\#friends)}}{\boldsymbol{200\sqrt{n}}}, \boldsymbol{1}\right)$$

Rumusan penghitung standar deviasi yakni:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

account	urltweet	urlredirect
AgusAdiWirawan	http://t.co/YixMZxmA2K	http://bit.ly/asistenPJK04
aldoalase	http://t.co/9L3SF4Q921	http://bit.ly/asistenPJK04
novitanataa	http://t.co/4POzUvuc0W	http://bit.ly/asistenPJK04
ebeckz22	http://t.co/BycuFML6aG	http://bit.ly/asistenPJK04
happyach	http://t.co/D9KJnTwqay	http://bit.ly/asistenPJK04

Gambar 5.36 Pengirim yang telah mengirimkan *entrypoint URL* yang sama

Contoh pengirim *entrypoint URL* yang ditunjukan pada Gambar 5.36, dimana mempunyai frekuensi *entrypoint URL* sejumlah lima.

Pada Gambar 5.37 merupakan jumlah *friend* tiap-tiap pengirim yang telah mengirimkan *entrypoint URL* yang sama. Sama dengan kasus sub bab sebelumnya, setelah mendapatkan jumlah *friend* dari masing-masing pengirim, didapatkan standar deviasi dengan mencari rata-rata dari kelima jumlah *friend* tersebut, dan dihitung varian dari masing-masing nilai tersebut.

Dari hasil perhitungan standar deviasi akan dilakukan pengecekkan, karena biasanya akun-akun palsu yang telah dibuat oleh penyerang mempunyai kemiripan nilai yang sama, oleh sebab itu apabila semakin mirip nilai atau jumlah *friend* yang dimiliki oleh akun-akun palsu, semakin mendekati nilai nol, namun apabila *entrypoint URL* yang dihasilkan hanya memiliki frekuensi satu, maka nilai standar deviasi yang dihasilkan akan bernilai nol. Untuk lebih jelasnya akan ditunjukan pada Gambar 5.37.

account	following
AgusAdiWirawan	307
aldoalase	372
novitanataa	173
ebeckz22	338
happyach	390

Gambar 5.37 Jumlah *friends* pada masing-masing pengirim *entrypoint URL*

Rata-rata: $\frac{307+372+173+338+390}{5} = \frac{1580}{5} = 316$

Varian masing – masing *followers* ditunjukan pada Tabel 5.9.

Tabel 5.9 nilai varian terhadap rata-rata yang diperoleh

friends	varian	hasil
307	$(307-316)^2$	81
372	$(372-316)^2$	3136
173	$(173-316)^2$	20449
338	$(338-316)^2$	484
390	$(390-316)^2$	5476
	jumlah	29626

Standar deviasi: $\sqrt{\frac{29626}{5}} = \sqrt{5925.2} = 76.97532$

Normalisasi: $\mathbf{min}(\frac{76.97532}{200\sqrt{5}}, 1) = 0.17212204972054$

Sehingga nilai normalisasi atas standar deviasi dari *friend* bernilai lebih besar dari nilai pada akun-akun palsu yang dihasilkan.

account	urltweet	SD_friends
AgusAdiWirawan	http://t.co/YixMZxmA2K	0.17212204972
aldoalase	http://t.co/9L3SF4Q921	0.17212204972
novitanataa	http://t.co/4POzUvuc0W	0.17212204972
ebeckz22	http://t.co/BycuFML6aG	0.17212204972
happyach	http://t.co/D9KJnTwqay	0.17212204972

Gambar 5.38 Nilai yang didapat pada simpangan baku *follower* setelah dilakukan normalisasi

Seperti yang ditunjukan pada Gambar 5.38, didapat nilai dari simpangan baku pada *friend* pengirim. Nilai tersebut didistribusikan kepada akun yang telah mengirimkan *entrypoint URL* yang sama.

5.2.6.9. Fitur Pengambilan Nilai Simpangan Baku Pada Rasio Dari *Friend-Follower* Pengirim

Selain melakukan uji coba terhadap simpangan baku *followers* dan *friends* juga dilakukan pengecekan kemiripan perbandingan antara *follower* dan *following*. Akun – akun palsu yang dibuat oleh penyerang biasanya mempunyai angka *friends* yang lebih besar dengan *followers*. Sehingga menggunakan perbandingan Antara *friends* dan *followers* yang biasanya mirip antara akun satu dengan akun yang lain. Pada uji coba proses pengambilan nilai simpangan baku pada rasio *friend-follower* pada pengirim yang telah mengirimkan *entrypoint URL* yang sama agar mengetahui jarak ukuran penyebaran perbandingan pada *follower* dan *friend* dari masing-masing pengirim. Sama seperti kasus pada sub bab sebelumnya, pada Gambar 5.39 merupakan jumlah dari masing-masing *friend-follower* tiap pengirim. Hal ini diperlukan untuk menghitung nilai rasio *friend-follower* agar mengetahui perbandingan antara *friend* dan *follower* tiap pengguna yang mengirimkan *entrypoint URL* yang sama.

account	follower	following
AgusAdiWirawan	174	307
aldoalase	377	372
novitanataa	243	173
ebeckz22	392	338
happyach	539	390

Gambar 5.39 Jumlah *friends-followers* pada masing-masing pengirim *entrypoint URL*

Dari yang ditunjukan pada Gambar 5.38,Untuk menghitung rasio dari *friend-follower* pengirim, dirumuskan menjadi berikut:

$$\frac{\boldsymbol{min}(\#\boldsymbol{followers}, \#\boldsymbol{following})}{\boldsymbol{max}(\#\boldsymbol{followers}, \#\boldsymbol{following})}$$

Dari masing – masing *following-follower* pengirim dicari nilai minimum dan maksimum dari masing – masing *friends* dan *followers*. Sehingga rasio yang didapat ditunjukan pada Tabel 5.10.

Tabel 5.10 nilai rasio *follower-friend* pengirim

follower	*friends*	Rasio	hasil
174	307	174/307	0.56677
377	372	372/377	0.98674
243	173	173/243	0.7119341
392	338	338/392	0.862244
539	390	390/539	0.7235621
		jumlah	3.85125025

Rata-rata: $\frac{3.85125025}{5}$ = 0.7702500

Varian masing – masing rasio *followers-friends* pada pengirim ditunjukan pada Tabel 5.11.

Tabel 5.11 nilai varian terhadap rata-rata yang diperoleh

Rasio	varian	hasil
0.56677	$(0.56677 - 0.7702500)^2$	0.041404
0.98674	$(0.98674 - 0.7702500)^2$	0.046868
0.7119341	$(0.7119341 - 0.7702500)^2$	0.003401
0.862244	$(0.862244 - 0.7702500)^2$	0.008463
0.7235621	$(0.7235621 - 0.7702500)^2$	0.00218
	jumlah	0.102315

Standar deviasi: $\sqrt{\frac{0.102315}{5}} = 0.143049$

Normalisasi: $\mathbf{min}(\frac{0.143049}{\sqrt{5}}, 1) = 0.063974$

Sehingga didapat nilai dari standar deviasi pada rasio dari *friend-follower* pada pengirim adalah 0.063974.

account	urltweet	SD_ratio_ff
AgusAdiWirawan	http://t.co/YixMZxmA2K	0.0639725956325018
aldoalase	http://t.co/9L3SF4Q921	0.0639725956325018
novitanataa	http://t.co/4POzUvuc0W	0.0639725956325018
ebeckz22	http://t.co/BycuFML6aG	0.0639725956325018
happyach	http://t.co/D9KJnTwqay	0.0639725956325018

Gambar 5.40 Nilai yang didapat pada simpangan baku pada rasio *friend-follower* setelah dilakukan normalisasi

Seperti yang ditunjukan pada Gambar 5.40. nilai simpangan baku pada rasio *friend-follower* telah dilakukan normalisasi. Hasil nilai yang didapat setelah itu didistribusikan kepada akun yang telah mengirimkan *entrypoint URL* yang sama.

5.2.6.10. Fitur Pengambilan Nilai Kemiripan Teks *Tweet* Pada Pengirim

Pada uji coba proses pengambilan nilai kemiripan teks *tweet* pada pengirim. Perlu diketahui bahwa biasanya, akun – akun palsu tersebut mendistribusikan *tweet*-nya dengan cara *spam* sehingga besar kemungkinan bahwa kemiripan teks pada *timeline* akun tersebut besar. Berbeda dengan akun-akun palsu yang dibuat oleh penyerang, karena akun Twitter yang asli biasanya menulis *tweet* yang sewajarnya. Oleh sebab itu, kemiripan teks didapat dengan menggunakan Jaccard *index*. Pada proses pengujian dilakukan dua kali proses pengujian, yakni pada akun @ITS_Surabaya, dan akun @mardiTA110. Untuk membuktikannya, pada uji coba pada kasus yang pertama, yakni pada akun @ITS_Surabaya dan ditunjukan pada Gambar 5.41 merupakan *tweet* yang muncul pada *timeline* pengguna dan Gambar 5.42 merupakan isi *timeline* pada @ITS_Surabaya, Normalisasinya adalah:

$$\sum_{t,u \in \textit{pasangan teks pada tweets}} \frac{\mathcal{J}(t,u)}{|\textit{pasangan teks pada tweets}|}$$

$$\frac{\mathcal{J}(t,u_1) + \mathcal{J}(t,u_2) + \mathcal{J}(t,u_3) + \cdots + \mathcal{J}(t,u_n)}{\sqrt{{u_1}^2 + {u_2}^2 + {u_3}^2 + \cdots + {u_n}^2}}$$

Dimana Jaccard *Index* merupakan:

$$\mathcal{J}(t,u) = \frac{|t \cap u|}{|t \cup u|}$$

account	tweet
ITS_Surabaya	ITS Surabaya Bersikap Netral dalam Pilpres http://...

Gambar 5.41 *Tweet* yang muncul pada *timeline* pengguna

Gambar 5.42 kumpulan *tweet* pada *timeline* akun @ITS_Surabaya

Setelah mendapatkan data *tweet* pada *timeline* akun @ITS_Surabaya, didapat Jaccard *index*, yakni ditunjukan pada Tabel 5.12.

Tabel 5.12 nilai varian terhadap rata-rata yang diperoleh

tweet	Jumlah kata	Jaccard *index*
1	7	1
2	10	0
3	2	0
4	7	0.166666667
5	4	0.083333333
6	6	0.083333333
7	6	0.083333333

$$\frac{1+0+0+0.1666667+0.08333+0.08333+0.08333}{\sqrt{7^2+10^2+2^2+7^2+4^2+6^2+6^2}} = 0.05079915450$$

Pada nilai kemiripan teks yang didapat, dilakukan pembulatan. Semakin besar nilai kemiripan teks, semakin mendekati kategori *spam*. dimana dapat ditunjukan pada Tabel 5.13.

Tabel 5.13 nilai varian terhadap rata-rata yang diperoleh

Teks similarity (TS)	pembulatan
TS < 0.1	0
0.1 < TS < 0.2	0.1
0.2 < TS < 0.3	0.2
0.3 < TS < 0.4	0.3
0.4 < TS < 0.5	0.4
0.5 < TS < 1	0.5

account	urltweet	text_similarity
ITS_Surabaya	http://t.co/eHVQHmIXuR	0

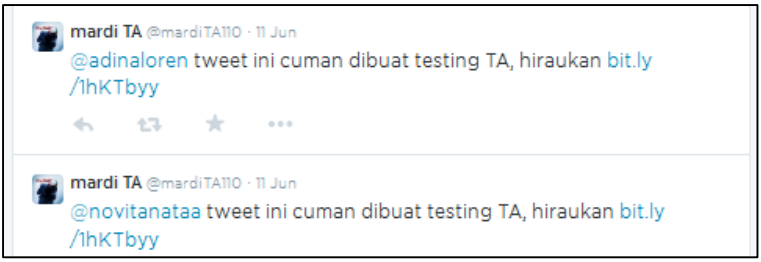
Gambar 5.43 Nilai dari kemiripan teks yang didapat pada akun @ITS_Surabaya

Pada Gambar 5.43 merupakan nilai dari pada Kemiripan teks. Uji coba kedua yakni pada *timeline* @mardiTA110, contoh akun yang bersifat spam. Hal ini ditunjukan pada Gambar 5.44 pada sisi *timeline* pengguna pengguna, dan Gambar 5.45 dan Gambar 5.46 pada sisi *timeline* akun pengirim tersebut.

account	tweet
mardiTA110	@novitanataa tweet ini cuman dibuat testing TA, hi...

Gambar 5.44 *Tweet* yang muncul pada *timeline* pengguna

Gambar 5.45 Kumpulan *tweet* pada *timeline* akun @mardiTA



Gambar 5.46 Kumpulan *tweet* pada *timeline* akun @mardiTA

Setelah mendapatkan data *tweet* pada *timeline* akun @mardiTA, didapat Jaccard *index*, yakni ditunjukan pada Tabel 5.14.

Tabel 5.14 nilai varian terhadap rata-rata yang diperoleh

tweet	Jumlah kata	Jaccard *index*
1	9	0.8
2	9	0.8
3	9	0.8
4	9	0.8
5	9	0.8
6	9	1

$$\frac{0.8+0.8+0.8+0.8+0.8+1}{\sqrt{8^2+8^2+8^2+8^2+8^2+8^2}} = 0.28818631259$$

account	urltweet	text_similarity
mardiTA110	http://t.co/sDUzdP93AX	0.2

Gambar 5.47 Nilai dari kemiripan teks yang dihasilkan

Pada Gambar 5.47, didapat nilai kemiripan tekk yang dihasilkan, dimana telah mengalami pembulatan. Perlu diketahu bahwa semakin besar nilai dari kemiripan teks, semakin besar dikategorikan akun tersebut bersifat *spam*.

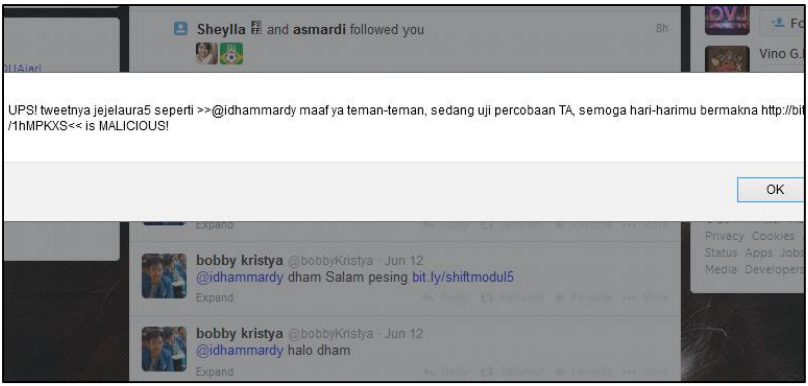
5.2.7. Melihat Notifikasi Peringatan Untuk *URL* yang Terdeteksi Pada Peramban Mozilla Firefox

Pada uji coba ini akan dilihat apakah sistem dapat memberikan informasi mengenai hasil dari *tweet* yang telah melalui proses ekstraksi fitur dan klasifikasi menggunakan algoritma *decision tree*. Hasil tersebut dikirim berupa *javascript* yang digunakan untuk memanipulasi tampilan pada Twitter dengan menggunakan Greasemonkey *add-ons* pada peramban Mozilla Firefox. Gambar 5.48 Notifikasi URL berbahaya pada Mozilla Firefoxmenunjukkan prosedur uji coba yang dilakukan pada proses melihat hasil dari *tweet* yang mengandung *URL* yang telah dideteksi.

Pada Gambar 5.48 Notifikasi URL berbahaya pada Mozilla Firefox, menunjukkan bahwa adanya *URL* yang berbahaya pada *timeline* ataupun *mention*. Dengan demikian tampilan peringatan pada *https://www.twitter.com* dapat bekerja. Untuk mengetahui keberhasilan pada uji coba notifikasi untuk *tweet* yang dideteksi dapat dilihat pada Tabel 5.15.

Tabel 5.15 Uji coba melihat notifikasi untuk *tweet* yang dideteksi

ID	**UJ-07**
Nama	Uji coba melihat *URL* yang telah dideteksi melalui peramban Mozilla Firefox
Tujuan Uji Coba	Menguji proses mendapatkan hasil
Kondisi Awal	Aplikasi berjalan, pengguna sudah *login*, pengecekan telah usai
Skenario 1	**Terdapat *URL* berbahaya**
Masukan	*Tweet* yang telah melalui proses korelasi dan pengecekkan dengan algoritma *decision tree*
Keluaran yang diharapkan	Aplikasi akan menampilkan peringatan pada peramban Mozilla Firefox
Hasil Uji Coba	BERHASIL



Gambar 5.48 Notifikasi URL berbahaya pada Mozilla Firefox

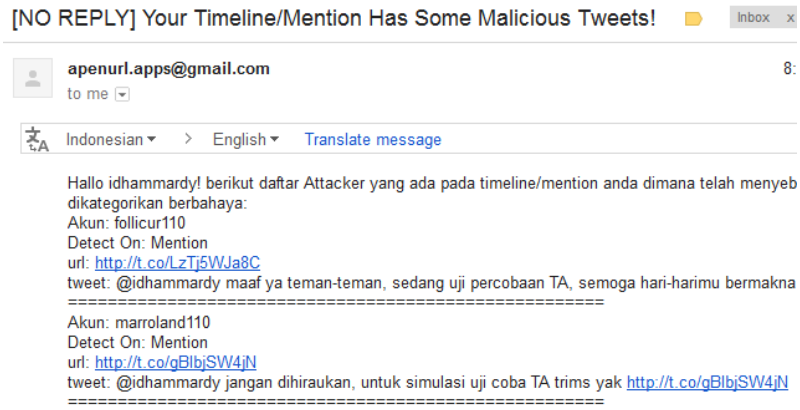
5.2.8. Menerima notifikasi *URL* berbahaya melalui *e-mail*

Tabel 5.16 Uji coba menerima notifikasi melalui *e-mail*

ID	**UJ-08**
Nama	Uji coba melihat rincian *tweet* dengan kondisi tidak normal melalui *e-mail*
Tujuan Uji Coba	Menguji proses mendapatkan informasi detil suatu tempat
Kondisi Awal	Aplikasi berjalan, pengguna sudah *login* dan sudah dilakukan pengecekan pada sistem
Skenario 1	**Sistem berhasil mengirimkan rincian *tweet* yang mengandung *URL* yang berbahaya melalui *e-mail***
Masukan	*URL* yang dideteksi berbahaya pada *tweet*
Keluaran yang diharapkan	Aplikasi menampilkan rincian informasi dari t*weet* yang dibajak melalui *e-mail*
Hasil Uji Coba	BERHASIL

Pada uji coba ini akan dilakukan proses pengecekan apakah memberikan notifikasi kepada pengguna melalui *e-mail* mengenai hal rincian *tweet* yang mengandung *URL* yang berbahaya. Sehingga ketika data *tweet* yang sudah dilabel. Tabel 5.16

menunjukkan prosedur uji coba yang dilakukan pada proses pengiriman *e-mail* pengguna. Gambar 5.49 Tampilan notifikasi melalui *e-mail* telah menunjukkan sistem berhasil mengirimkan rincian pada *e-mail* tersebut berisi akun, dideteksi pada *timeline* atau *mention*.



Gambar 5.49 Tampilan notifikasi melalui *e-mail*

5.3. Uji Coba Kasus

Pada pengujian kasus, akan dijelaskan kasus yang mungkin akan dihadapi sistem. Uji coba yang akan dilakukan adalah pencarian model pada metode klasifikasi *decision tree* dimana akan dijelaskan pada subbab berikut.

5.3.1. Uji Coba Pemodelan *Decision Tree*

Dalam mengambil keputusan ada tidaknya *URL* berbahaya pada *timeline* dan *mention* pengguna, perlu dilakukan klasifikasi pada setiap *URL* yang dikirimkan pada pengguna melalui *tweet*. Metode klasifikasi yang digunakan pada sistem ini adalah *decision tree*, karena metode klasifikasi ini merupakan metode pengambilan keputusan tercepat dibandingkan metode-metode yang sudah ada. Pada uji coba ini dilakukan lima pemodelan *tree* yang digunakan untuk data *training*. Hal ini bertujuan untuk mencari model terbaik

dari sekumpulan data yang telah diambil. Sehingga dari kelima model tersebut diambil dengan jumlah data yang berbeda-beda. Kelima model tersebut diperoleh dari pengambilan *tweet* random yang diambil berdasaran fungsi *search* pada Twitter. Sehingga dengan fungsi *search* pada Twitter mengambil beberapa kasus layanan *shorthening URL Service* seperti *http://bit.ly*, *http://tinyurl.com/*, dan juga *http://ow.ly/*. Dengan layanan tersebut, memunkinkan adanya *tweet* yang mengandung *URL* yang terindikasi berbahaya.

Dalam pembuatan pemodelan *tree*, yakni pemodelan A mempunyai jumlah data sebanyak 100 data, pemodelan B mempunyai jumlah data sebanyak 200 data, pemodelan C mempunyai jumlah data sebanyak 300 data, pemodelan D mempunyai jumlah data sebanyak 500 data, dan pemodelan E mempunyai data sebanyak 1000 data. Masing-masing data tersebut diambil secara sekuensial dan bertahap dari 100 hingga 1000 data. Setelah didapat jumlah data pada masing-masing pemodelan, dilakukan pelabelan secara manual untuk mengidentifikasikan berbahaya tidaknya *URL* yang telah dicek. Tabel 5.17 menunjukkan masing-masing pemodelan beserta akurasi yang didapat dengan menggunakan 10 *cross validation*. Untuk masing-masing model dapat dilihat pada Lampiran R.

Tabel 5.17 Uji coba akurasi pemodelan menggunakan 10 *cross validation*

Data Training ke-	Nama model	Jumlah Data	Akurasi (%)
1	A	100	95.5556
2	B	200	96.8586
3	C	300	97.931
4	D	500	99.5984
5	E	1000	99.5992
		Rata-rata	98.11356

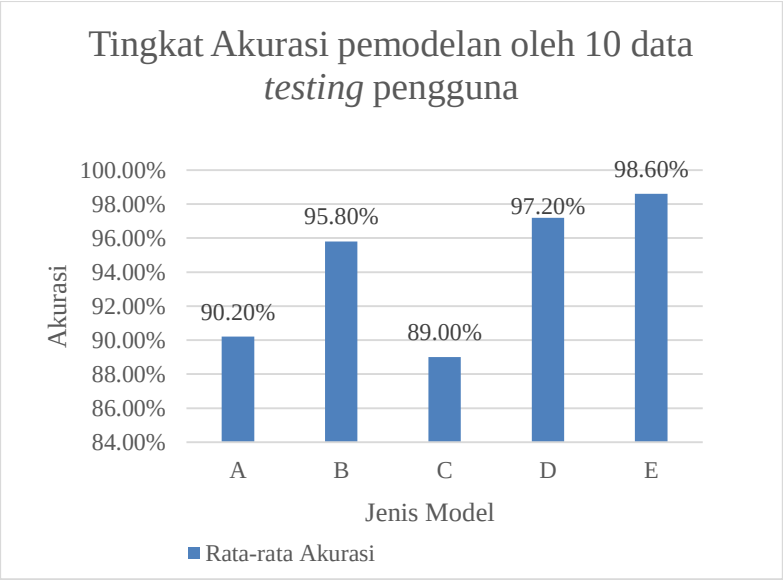
Pemodelan *tree* yang didapat membangun *tree* yang berbeda-beda disetiap data *training*. Semakin banyaknya jumlah data yang dijadikan data *training*, semakin kompleks pemodelan *tree* yang dibangun. Proses pengujian penentuan model yang terbaik dilakukan dengan menguji masing-masing model tersebut dengan beberapa pengguna. Dalam hal ini sistem menguji akurasi pada 10 pengguna dengan data yang berbeda. hal ini bertujuan untuk mengukur tingat akurasi yang didapat. Karena semakin banyak data uji yang dilakukan maka semakin besar juga tingkat akurasi yang didapat dari masing-masing model. Pemodelan terbaik dipilih berdasarkan nilai rata-rata akurasi terbesar pada masing-masing pengguna yang dijadikan data *testing*. Berdasarkan hasil perhitungan akurasi yang dilakukan dengan menggunakan WEKA, didapatkan kesimpulan bahwa model E dimana berisi 1000 data memiliki tingkat akurasi terbesar dibandingkan dengan pemodelan yang lain. Hasil pengujian ini dapat dilihat pada Tabel 5.18.

Tabel 5.18 Uji coba akurasi pemodelan

	Akurasi Model (%)				
Pengguna	**A**	**B**	**C**	**D**	**E**
1	96	94	90	100	100
2	100	84	74	94	100
3	84	90	80	94	100
4	92	96	92	100	96
5	100	98	98	100	100
6	100	100	96	96	98
7	48	100	80	94	96
8	90	100	90	100	100
9	92	100	100	100	98
10	100	96	90	94	98
Rata-rata	90.2	95.8	89	97.2	98.6

Dari data yang didapatkan melalui pengujian pada Tabel 5.18, dapat disimpulkan, dengan menggunakan metode *decision tree* pada jumlah data yang bervariasi mempunyai nilai akurasi yang

berbeda ketika diuji oleh beberapa pengguna. Nilai akurasi terbesar diperoleh pada model E dengan jumlah data *training* 1000. Bisa dipastikan bahwa jumlah data *training* mempengaruhi rata-rata nilai akurasi yang diperoleh setelah dilakukan uji data *testing* ke beberapa pengguna. Gambar 5.50 merupakan grafik perbedaan rata-rata nilai akurasi yang didapat berdasarkan jumlah data pada data *training*.



Gambar 5.50 Grafik perbedaan tingkat akurasi berdasarkan model *tree*

Seperti yang ditunjukan pada Gambar 5.50 dijelaskan bahwa model A mempunyai tingkat akuraasi sebesar 90.2% setelah dilakukannya pengujian terhadap 10 data *testing* pengguna. sedangkan model E mempunyai tingkat akurasi sebesar 98.6% setelah dilakukannya pengujian terhadap 10 data *testing* pengguna.

5.4. Uji Coba Peforma

Pengujianperforma pada aplikasi ini dibagi menjadi beberapa bagian yang akan diterangkan pada sub bab berikut.

5.4.1. Uji Coba Waktu Kerja Proses

Uji waktu respon dilakukan untuk mengetahui waktu proses yang dibutuhkan dalam menjalankan sistem pendeteksi *URL* berbahaya pada Twitter.

Dalam uji coba kali ini dilakukan perhitungan waktu pada serangkaian alur kerja proses sistem. Terdapat delapan proses yang dilakukan untuk menjalankan proses sistem pendeteksi *URL* berbahaya. Setiap proses akan dihitung waktu respon yang dilakukan. Dalam uji coba ini dilakukan beberapa kali pengambilan data. Yakni pada 50 data, 100 data, 200 data, 300 data 500 data, dan 1000 data. Proses yang dihitung antara lain:

1. Proses pengambilan *tweet*
2. Proses pencarian *entrypoint URL*
3. Pengambilan *friend* dan *follower* pada pengirim
4. Pengambilan *tweet* pada *timeline* pengirim
5. Proses ekstraksi fitur
6. Klasifikasi metode *decision tree*
7. Notifikasi peringatan pada Mozilla Firefox
8. Pengiriman notifikasi melalui *e-mail*

Pada Gambar 5.51 menunjukan grafik diagram waktu respon yang dilakukan pada 50 *tweet*, 100 *tweet*, 200 *tweet*, 300 *tweet*, 400 *tweet*, dan 500 *tweet*. Hasil pengujian waktu kerja proses pada sistem ditunjukan pada Tabel 5.19

Pada proses ekstraksi fitur, terdapat 10 fitur yang yang diambil setelah dapat melakukan proses ekstraksi rantai *redirect URL* dan ekstraksi konteks informasi *tweet* pada pengirim. Setiap pengambilan nilai pada masing–masing fitur dihitung waktu proses pada setiap pengambilan data. Fitur-fitur yang diuji waktu prosesnya antara lain:

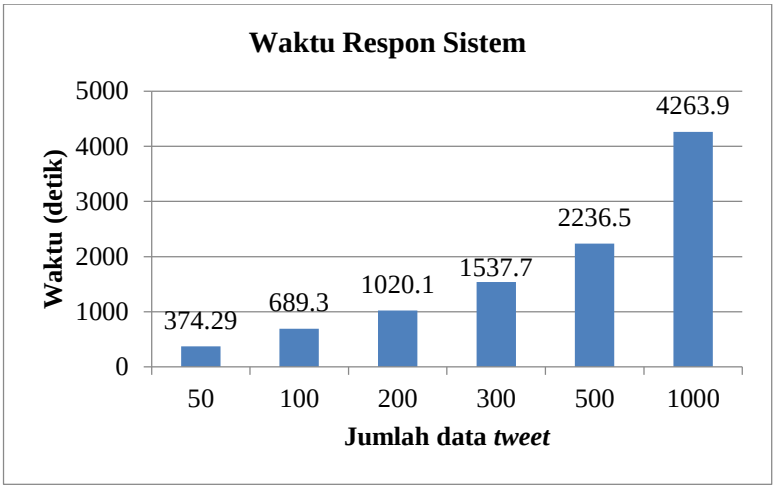
a. Proses pengambilan nilai panjang rantai *redirect URL*
b. Proses pengambilan jumlah frekuensi *entrypoint URL*

c. Proses pengambilan nilai letak *entrypoint URL*
d. Proses pengambilan jumlah inisial *URL* pada *entrypoint URL* yang sama
e. Proses pengambilan nilai perbedaan *landing URL* pada *entrypoint URL* yang sama
f. Proses pengambilan jumlah pengirim *entrypoint URL*
g. Proses pengambilan nilai simpangan baku pada *followers* pengirim *entrypoint URL*
h. Proses pengambilan nilai simpangan baku pada *friends* pengirim *entrypoint URL*
i. Proses pengambilan nilai simpangan baku pada rasio dari *friends* dan *followers* pengirim *entrypoint URL*
j. Proses pengambilan nilai kemiripan teks pada *timeline* pengirim.

Tabel 5.19 Pengujian waktu respon proses

Proses	**Waktu kerja berdasarkan data (detik)**					
	50	**100**	**200**	**300**	**500**	**1000**
1	47.603	97.71	162.47	207.59	415.64	828.5
2	0.17	0.322	0.6254	1.099	1.5351	3.277
3	0.24	0.404	0.808	1.212	2.02	4.04
4	18.05	36.1	72.2	108.3	180.5	361
5	308.20	549.5	778.3	1213.7	1631.1	3060.3
6	0.0014	0.002	0.0026	0.0031	0.0046	0.0054
7	0.0193	0.003	0.0042	0.0047	0.0055	0.0064
8	0.0009	5.307	5.67	5.87	5.74	6.73
Total	374.29	689.3	1020.1	1537.7	2236.5	4263.9

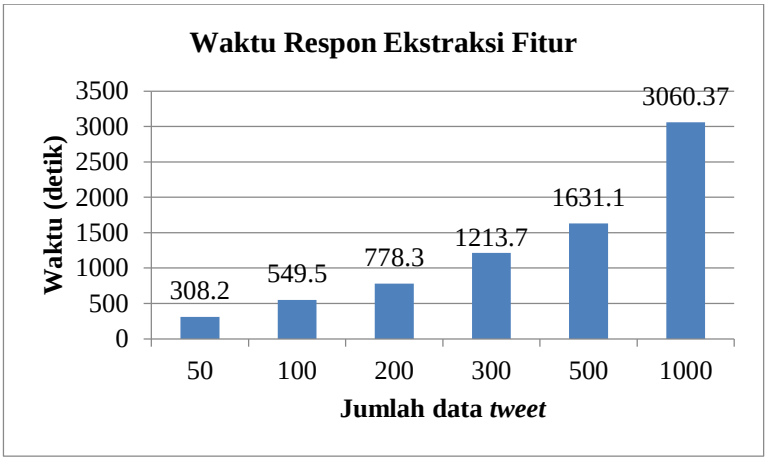
Pada Gambar 5.52 menunjukan grafik waktu respon pengambilan ekstaksi fitur yang dilakukan pada 50 *tweet*, 100 *tweet*, 200 *tweet*, 300 *tweet*, 400 *tweet*, dan 500 *tweet*. Hasil pengujian waktu kerja proses pada masing-masing fitur ditunjukan pada Tabel 5.20.



Gambar 5.51 Waktu respon pada jumlah data *tweet* yang berbeda

Tabel 5.20 Pengujian waktu respon ekstraksi fitur

Proses	Waktu kerja berdasarkan data (detik)					
	50	**100**	**200**	**300**	**500**	**1000**
a	1.1834	2.4802	3.736	6.0155	9.7025	17.10
b	2.77	6.208	7.860	11.81	18.843	35.81
c	1.1835	2.4804	3.716	6.0135	9.6825	17.10
d	2.008	4.204	7.260	11.815	16.843	35.81
e	294.04	521.43	734.6	1147.2	1507.8	2815.1
f	1.387	2.213	4.031	6.267	14.049	31.009
g	1.532	2.7001	4.531	6.667	16.054	32.492
h	1.532	2.7001	4.531	6.667	16.054	32.492
i	1.532	2.7001	4.531	6.667	16.054	32.492
j	1.0252	2.423	3.542	4.563	5.98	10.89
Total	308.20	549.5	778.3	1213.7	1631.1	3060.37



Gambar 5.52 Waktu respon ekstraksi fitur pada jumlah data *tweet* yang berbeda

5.5. Evaluasi Uji Coba

Dari hasil uji coba fungsional yang telah dilakukan terhadap aplikasi ini, maka dapat dibuat kesimpulan keberhasilan uji coba yang ditunjukkan oleh Tabel 5.21.

Tabel 5.21 Evaluasi uji coba fungsionalitas

No	Nama Proses	Hasil	Catatan
1	pengambilan *tweet*	Berhasil	-
2	pencarian *entrypoint URL*	Berhasil	-
3	Pengambilan *friend* dan *follower* pada pengirim	Berhasil	-
4	Pengambilan *tweet* pada *timeline* pengirim	Berhasil	-
5	Ekstraksi fitur	Berhasil	-
6	Klasifikasi metode *decision tree*	Berhasil	-
7	peringatan pada Mozilla Firefox	Berhasil	-
8	Pengiriman notifikasi melalui *e-mail*	Berhasil	-

Tabel 5.21 menunjukkan bahwa antara desain perancangan sistem, implementasi, serta uji coba yang telah dilakukan telah sesuai dan dapat berjalan dengan baik yang dibuktikan dengan keberhasilan semua uji coba yang telah dilakukan.

Sistem juga telah melewati uji coba kasus yaitu pengujian kemampuan sistem dalam ketepatan dan akurasi pemberian kesimpulan terhadap *URL* yang diperoses, dengan menggunakan metode klasifikasi *decision tree* pada pemodelan yang terbentuk pada 1000 data *training* dengan akurasi 99.59%. pemodelan tersebut mampu memberikan ketepatan akurasi pada 10 data *testing* sebesar 98.6%.

Dari sisi uji coba peforma sistem juga mampu menunjukan hasil rata-rata waktu 374.29 detik dalam proses pengecekkan 50 data, 689.3 detik pada 100 data, 1020.1 detik pada 200 data, 1537.7 detik pada 300 data, 2236.5 detik pada 500 data, 4263.9 detik pada 1000 data. Hal ini menunjukan bahwa proses pendeteksian *URL* pada *tweet* memerlukan waktu yang cukup lama pada pengambilan data tertentu.

.

BAB VI
PENUTUP

Pada bab terakhir ini akan dibahas mengenai kesimpulan yang diperoleh selama pengerjaan Tugas Akhir hingga selesai. Pada bab ini juga akan menjawab pertanyaan yang dijabarkan pada BAB I. Untuk memperbaiki semua kelebihan dan kekurangan dari sistem, akan dijelaskan pada subbab saran.

6.1. Kesimpulan

Berikut adalah beberapa kesimpulan yang diperoleh dari proses pengerjaan Tugas Akhir ini:

1. Sistem dapat mendeteksi adanya suatu *URL* berbahaya yang mempunyai penjang rantai *redirect URL* dan mempunyai lebih dari satu frekuensi *entrypoint URL* pada pengecekan sejumlah *tweet* serta mempunyai perbedaan *landing URL* ketika dilakukan pengecekan pada *UserAgent* yang berbeda. *URL* tersebut juga didistribusikan oleh akun-akun palsu yang mempunyai kemiripan *tweet* pada isi *timeline* mereka, dan mempunyai kemiripan yang besar terhadap jumlah *follower* dan *following* akun palsu yang lain.
2. Penggunaan algoritma *Decision Tree* mengimplementasikan bahwa adanya seleksi fitur yang terdapat pada sistem. Dimana hanya terdapat lima fitur yang dihasilkan pada pohon keputusan (*decision tree*). Yakni fitur nilai perbedaan *landing URL*, letak pada *entrypoint URL*, jumlah inisial *URL*, nilai kemiripan teks *tweet*, dan juga nilai simpangan baku pada *followers*. Hal itu didasari karena kurang beragamnya data *training* yang didapatkan dari pengambilan jumlah *tweet*.
3. Berdasarkan hasil uji coba, pengambilan data *training* pada *tweet* ikut mempengaruhi penentuan pemodelan pohon keputusan dengan tingkat akurasi yang berbeda-beda. Setelah dilakukan pengujian pada data *testing* sejumlah 10 pengguna yang berbeda-beda mempunyai tingkat akurasi dengan nilai

98.6% pada pemodelan pohon keputusan dengan jumlah data *training* 1000 data.
4. Waktu respon yang diperoleh dari masing-masing proses berpengaruh dari jumlah data *tweet* yang diambil dan dilakukan pengecekkan. Dari Hasil pengujian untuk waktu respon pada 1000 data pengecekkan adalah 3060.37 detik, sedangkan untuk waktu respon pada 50 data pengecekkan hanya membutuhkan waktu 308.20 detik.

6.2. Saran

Adapun saran dari penulis yang dapat diberikan dari kekurangan sistem yang ada, maupun pengembangan lebih lanjut yang dapat dilakukan yaitu:

1. Pengembangan sumber daya sistem baik dari sisi perangkat lunak maupun perangkat keras akan sangat dibutuhkan ketika jumlah pengguna dan kebutuhan sistem semakin meningkat.
2. Penggunaan *library* untuk mengembangkan proses pengolahan data pada Twitter mempengaruhi proses analisis *behavioral* pada *URL*. Kebutuhan akan penggunaan *library* yang mampu memiliki *limit* lebih besar akan sangat membantu dalam proses tersebut.
3. Kebutuhan koneksi Internet yang cukup stabil merupakan salah satu permasalahan yang terjadi dalam sistem pendeteksi, terutama dalam proses pengumpulan data.
4. Penambahan jumlah data *training* untuk meningkatkan akurasi deteksi aplikasi

DAFTAR PUSTAKA

[1] S. L. a. J. Kim, "Warning Bird: Detecting Suspicious URLs in Twitter Stream," *POSTECH Pohang University of Science and Technology,* pp. 1-13, 2011.

[2] MathIsFun, "Math Is Fun Advanced - Standart Deviation Formula," [Online]. Available: http://www.mathsisfun.com/data/standard-deviation-formulas.html. [Accessed 2014].

[3] T. Inc, "Rules and Best Practices," 2014. [Online]. Available: https://support.twitter.com/articles/31796-my-account-has-been-compromised.

[4] Q. C. Headquarters, "Webopedia What is URL?," [Online]. Available: http://www.webopedia.com/TERM/U/URL.html.

[5] Timothy-Makarov, "Github Tweetsharp Twitter API," 2014. [Online]. Available: https://github.com/timothy-makarov/tweetsharp.

[6] W. I. M. I. Address, "what is the user-agent," [Online]. Available: http://whatismyipaddress.com/user-agent. [Accessed 2014].

[7] Google, "Google Crawlers," Google, [Online]. Available: https://support.google.com/webmasters/answer/1061943?hl=en. [Accessed 2014].

[8] M. Rouse, "WhatIs.com Greasemonkey add-ons," March 2007. [Online]. Available: http://whatis.techtarget.com/definition/Greasemonkey.

[9] M. Foundation, "About Javascript," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript. [Accessed 2014].

[10] J. Quinlan, "Introduction of Decision Tree," *Machine Learning*, vol. 1, pp. 81-106, 1986.

[11] W. University, "WEKA - Machine Learning Groups," Waikato University, [Online]. Available: http://www.cs.waikato.ac.nz/ml/index.html. [Accessed 2014].

[12] B. A. Minartiningtyas, "Informatika - Definisi dan Tujuan Normalisasi," [Online]. Available: http://informatika.web.id/definisi-dan-tujuan-normalisasi.htm. [Accessed 2014].

[13] Code10, "Jaccard Similarity," [Online]. Available: http://www.code10.info/index.php?article_jaccard-similarity. [Accessed 2014].

LAMPIRAN

Bagian ini merupakan lampiran sebagai dokumen pelengkap dari buku Tugas Akhir dimana akan diberikan potongan kode sumber dari fungsi-fungsi yang digunakan untuk membangun sistem.

A. Login dan Otorisasi Twitter

Proses pertama yang dilakukan ketika menjalankan aplikasi *client* adalah proses *login* halaman Twitter pada *default browser* pengguna untuk mendapatkan kode otorisasi. Kode sumber ini ditunjukkan pada Kode Sumber 1 Login dan otorisasi pada Twitter Kode Sumber 2 dan Kode Sumber 3.

```
try
{
    verifier = txtAuth.Text;
    twit.request(verifier);
    pengguna.cekUser();
    bool flag = true;
    if (variables.v_itungUser == 0)
    {
        MessageBox.Show("Selamat Datang
"+variables.v_username+"! terima kasih sudah
mendaftar aplikasi ini!");
    }
    else if (variables.v_itungUser != 0)
    {
        MessageBox.Show("halo, " +
variables.v_username + "!");
        }
    }
catch
    {
        MessageBox.Show("Authorization Failed!
          Please check your connection!");
    }
```

Kode Sumber 1 Login dan otorisasi pada Twitter

```
twit.getProfiles();
basdat.hitung_user();
if (variables.v_itungUser == 0)
{
    basdat.insert_user();
    buatUser();
}
else if (variables.v_itungUser != 0)
{
     variables.v_usertraining =
variables.v_username + "_";
}
```

Kode Sumber 2 Cek User

```
TwitterUser tweetuser = service.GetUserProfile(new
GetUserProfileOptions()
{
      IncludeEntities = false,
      SkipStatus = false
});

            variables.v_userid =
tweetuser.Id.ToString();
            variables.v_username =
tweetuser.ScreenName;
```

Kode Sumber 3 getProfiles

B. Proses Pengambilan *Tweet*

Pada proses ini aplikasi mengambil data *tweet* pengguna pada *timeline* dan *mention*. Kode sumber ini dapat dilihat pada Kode Sumber 4. Sistem akan terus melakukan perulangan hingga jumlah data *tweet* yang diinginkan.

```
var tweets = service.ListTweetsOnHomeTimeline(new
ListTweetsOnHomeTimelineOptions { Count = 70 });
            var mentions =
service.ListTweetsMentioningMe(new
ListTweetsMentioningMeOptions { Count = 100 });

            if (tweets != null)
            {
```

```
basdat.hitung_timeline(variables.v_typeTL);
                if (variables.v_itungtweet == 0)
                {
                    foreach (TwitterStatus tweet in
tweets)
                    {
                        TimeZoneInfo cstZone =
TimeZoneInfo.FindSystemTimeZoneById("SE Asia
Standard Time");
                        DateTime cstTime =
TimeZoneInfo.ConvertTimeFromUtc(tweet.CreatedDate,
cstZone);
                        foreach (var tweeturl in
tweet.Entities.Urls)
                        {
                            variables.v_urltweet =
tweeturl.Value;

variables.v_userScreenname = tweet.User.ScreenName;
                            variables.v_tweettext =
tweet.Text;
                            variables.v_cstTime =
cstTime;
                            variables.v_friends =
tweet.User.FriendsCount;
                            variables.v_followers =
tweet.User.FollowersCount;

variables.v_userScreenid =
tweet.User.Id.ToString();
                            variables.v_entrypoint
= null;


basdat.insert_table_username(variables.v_typeTL);

basdat.insert_table_usertraining(variables.v_typeTL
);

                            DateTime startCekAkun =
DateTime.Now;
                            basdat.cekAccount();
```

```
                                DateTime endCekAkun =
DateTime.Now;
                                double selisihCekAkun =
endCekAkun.Subtract(startCekAkun).TotalSeconds;

Console.WriteLine("waktu cek akun:" +
selisihCekAkun.ToString());

                                variables.url2 = new
List<string>();


variables.url2.Add(variables.v_urltweet);

                                peramban.RedirectURL();

urldetect.redirListURL();

                                DateTime
startSimilarity = DateTime.Now;

similar.SimilarityAccount();
                                DateTime endSimilairity
= DateTime.Now;
                                double
selisihSimilarity =
endSimilairity.Subtract(startSimilarity).TotalSecon
ds;

Console.WriteLine("waktu cek similar:" +
selisihSimilarity.ToString());
                            }
                        }
                    }
                    else if (variables.v_itungtweet >
0)
                    {

basdat.select_maxdate(variables.v_typeTL);

                        foreach (TwitterStatus tweet in
tweets)
                        {
```

```
                        TimeZoneInfo cstZone =
TimeZoneInfo.FindSystemTimeZoneById("SE Asia
Standard Time");
                        DateTime csTime =
TimeZoneInfo.ConvertTimeFromUtc(tweet.CreatedDate,
cstZone);

                        variables.v_cstTime =
csTime;


basdat.select_bandingTanggal();

                        if
(variables.hasilSelectDate > 0)
                        {
                            foreach (var tweeturl
in tweet.Entities.Urls)
                            {

variables.v_urltweet = tweeturl.Value;

variables.v_userScreenname = tweet.User.ScreenName;

variables.v_tweettext = tweet.Text;
                                variables.v_friends
= tweet.User.FriendsCount;

variables.v_followers = tweet.User.FollowersCount;

variables.v_entrypoint = null;


basdat.insert_table_username(variables.v_typeTL);

basdat.insert_table_usertraining(variables.v_typeTL
);

                                DateTime
startCekAkun = DateTime.Now;

basdat.cekAccount();
                                DateTime endCekAkun
= DateTime.Now;
```

```
                                    double
selisihCekAkun =
endCekAkun.Subtract(startCekAkun).TotalSeconds;

Console.WriteLine("waktu cek akun:" +
selisihCekAkun.ToString());

                                    variables.url2 =
new List<string>();


variables.url2.Add(variables.v_urltweet);


peramban.RedirectURL();

urldetect.redirListURL();

                                    DateTime
startSimilarity = DateTime.Now;

similar.SimilarityAccount();
                                    DateTime
endSimilairity = DateTime.Now;
                                    double
selisihSimilarity =
endSimilairity.Subtract(startSimilarity).TotalSecon
ds;

Console.WriteLine("waktu cek similar:" +
selisihSimilarity.ToString());
                                }
                            }
                        }
                    }
                }
                //mention
                if (mentions != null)
                {

basdat.hitung_timeline(variables.v_typeMention);

                    if (variables.v_itungtweet == 0)
                    {
```

```
                    foreach (TwitterStatus mention
in mentions)
                    {
                        TimeZoneInfo cstZone =
TimeZoneInfo.FindSystemTimeZoneById("SE Asia
Standard Time");
                        DateTime cstTime =
TimeZoneInfo.ConvertTimeFromUtc(mention.CreatedDate
, cstZone);
                        foreach (var tweeturl in
mention.Entities.Urls)
                        {
                            variables.v_urltweet =
tweeturl.Value;

variables.v_userScreenname =
mention.User.ScreenName;
variables.v_tweettext = mention.Text;
variables.v_cstTime = cstTime;
variables.v_friends = mention.User.FriendsCount;
variables.v_followers =
mention.User.FollowersCount;
variables.v_entrypoint = null;


basdat.insert_table_username(variables.v_typeMentio
n);

basdat.insert_table_usertraining(variables.v_typeMe
ntion);

                            DateTime startCekAkun =
DateTime.Now;
                            basdat.cekAccount();
                            DateTime endCekAkun =
DateTime.Now;
                            double selisihCekAkun =
endCekAkun.Subtract(startCekAkun).TotalSeconds;

Console.WriteLine("waktu cek akun:" +
selisihCekAkun.ToString());

                            //variables.url1 = new
List<string>();
```

```
                                variables.url2 = new
List<string>();


//variables.url1.Add(variables.v_urltweet);

variables.url2.Add(variables.v_urltweet);

                                peramban.RedirectURL();

urldetect.redirListURL();

                                DateTime
startSimilarity = DateTime.Now;

similar.SimilarityAccount();
                                DateTime endSimilairity
= DateTime.Now;
                                double
selisihSimilarity =
endSimilairity.Subtract(startSimilarity).TotalSecon
ds;

Console.WriteLine("waktu cek similar:" +
selisihSimilarity.ToString());
                            }
                        }
                    }
                    else if (variables.v_itungtweet >
0)
                    {

basdat.select_maxdate(variables.v_typeMention);
                        foreach (TwitterStatus mention
in mentions)
                        {
                            TimeZoneInfo cstZone =
TimeZoneInfo.FindSystemTimeZoneById("SE Asia
Standard Time");
                            DateTime csTime =
TimeZoneInfo.ConvertTimeFromUtc(mention.CreatedDate
, cstZone);
```

```
                        variables.v_cstTime =
csTime;


basdat.select_bandingTanggal();

                        if
(variables.hasilSelectDate > 0)
                        {
                            foreach (var tweeturl
in mention.Entities.Urls)
                            {

variables.v_urltweet = tweeturl.Value;

variables.v_userScreenname =
mention.User.ScreenName;

variables.v_tweettext = mention.Text;
                                variables.v_friends
= mention.User.FriendsCount;

variables.v_followers =
mention.User.FollowersCount;

variables.v_entrypoint = null;


basdat.insert_table_username(variables.v_typeMentio
n);

basdat.insert_table_usertraining(variables.v_typeMe
ntion);

                                DateTime
startCekAkun = DateTime.Now;

basdat.cekAccount();
                                DateTime endCekAkun
= DateTime.Now;
                                double
selisihCekAkun =
endCekAkun.Subtract(startCekAkun).TotalSeconds;
```

```
Console.WriteLine("waktu cek akun:" +
selisihCekAkun.ToString());

                                    variables.url2 =
new List<string>();


variables.url2.Add(variables.v_urltweet);


peramban.RedirectURL();

urldetect.redirListURL();

                                    DateTime
startSimilarity = DateTime.Now;

similar.SimilarityAccount();
                                    DateTime
endSimilairity = DateTime.Now;
                                    double
selisihSimilarity =
endSimilairity.Subtract(startSimilarity).TotalSecon
ds;

Console.WriteLine("waktu cek similar:" +
selisihSimilarity.ToString());
                                }
                            }
                        }
                    }
                }
```

Kode Sumber 4 Kode sumber pengambilan data *tweet*

C. Proses Ekstraksi Korelasi Rantai *Redirect URL*

Untuk melakukan ekstraksi pada korelasi rantai *redirect* URL dilakukan pencarian *entrypoint URL* dan mengetahui pengalihan *URL* pada rantai *redirect URL* terlebih dahulu. Agar mendapatkan lima fitur yang diinginkan yakni mengenai Kode sumber ini ditunjukkan pada Kode Sumber 5.

```
variables.EP_points = new List<string>();
            variables.EP_freq = new List<double>();
            Dictionary<int, List<string>>
redirectDict = new Dictionary<int,List<string>>();
            redirectDict= new Dictionary<int,
List<string>>();
            //List<string>kolom= new
List<string>();
            int indeks = 0;
            foreach(var s in variables.kolom)
            {
                redirectDict.Add(indeks,
ambilpath(s));
                indeks++;
            }
            double flag = 0;
            double freq_EP = 0;
            addToDict("sjdja", 2);
            foreach (var indeksDict in
redirectDict) //indeks barisnya
            {
                string entrypoint ="";
                string currentEP = "";
                flag = 1;
                for(int a = 0; a <
indeksDict.Value.Count; a++)
                //foreach (var isi in
indeksDict.Value) //perulangan ke kanan.
//valuedict // yang pengecek
                {

//MessageBox.Show(indeksDict.Value[a]);
                    freq_EP = 1;

                    //---------mengecek apabila
entrypoint baris ke-n sudah pernah dicek
                    foreach (var d in sameString)
                    {
                        if (d.Key ==
indeksDict.Value[a])
                        {
                            freq_EP = d.Value;
                            entrypoint =
indeksDict.Value[a];
```

```
                            currentEP = entrypoint;
                            flag = freq_EP;
                        }
                    }
                    if (freq_EP > 1)
                        continue;
                    //------------------------------
-------------------

                    //------------------------------
cek dengan indeks selanjutnya
                    for (int i = indeksDict.Key +
1; i <redirectDict.Count; i++) //yang dicek..
indeks tweet selanjutnya
                    {
                        foreach (string cmpdata in
redirectDict[i].ToArray()) //yg dicek //kekanan
                        {
                            if (indeksDict.Value[a]
== cmpdata)
                            {
                                //ini entrypoint
                                freq_EP++;
                                entrypoint =
indeksDict.Value[a];

//MessageBox.Show(entrypoint+"-"+freq_EP);
                            }
                        }

                    }
                    //------------------------------
-------



                    //if (flag > 1 && flag ==
freq_EP)
                    //    continue;
                    //if(flag==1 && flag ==
freq_EP)
                    entrypoint =
indeksDict.Value[0];
```

```
                    //dalam baris terdapat
frekuensi yg lebih besar, diset entrypoint dan
frekuensinya
                    if (flag < freq_EP)
                    {
                        flag = freq_EP;
                        entrypoint =
indeksDict.Value[a];
                        currentEP = entrypoint;

//MessageBox.Show(entrypoint);
                        addToDict(entrypoint,
freq_EP);
                    }

//MessageBox.Show(indeksDict.Value[a]+"-"+
entrypoint+"-"+flag+"-"+freq_EP);

                }
                //----kalo flag > 1 diset currentEP
                if (flag > 1)
                    entrypoint = currentEP;

                //

variables.EP_points.Add(entrypoint);
                variables.EP_freq.Add(flag);
            }
```

Kode Sumber 5 Mendapatkan *entrypoint URL*

D. Proses Ekstraksi Konteks Informasi *Tweet* Pada Pengirim

Proses ekstraksi konteks informasi *tweet* pada pengirim ini diambil untuk menentukan mendapatkan nilai dari fitur yang dihasilkan, seperti mendapatkan nilai standar deviasi antara *friend* dan *follower*, dan juga kemiripan *tweet* pada isi *timeline* pengirim. Kode sumber untuk mengambil *friend* dan *follower* pengirim ditunjukan pada Kode Sumber 6.

```
string akuns = "SELECT * FROM usertwitter WHERE
account='" + variables.v_userScreenname + "'";
            MySqlDataReader dr;
            con.Open();
```

```
            cmd = con.CreateCommand();
            cmd.CommandText = akuns;
            dr = cmd.ExecuteReader();
            if (dr.HasRows == false)
            {
                dr.Close();
                cmd.CommandText = "INSERT INTO
usertwitter(`user-id`,`account`, `following`,
`follower`)VALUES(?id,?akun,?following,?follower);"
;
                cmd.Parameters.AddWithValue("?id",
variables.v_userScreenid);

cmd.Parameters.AddWithValue("?akun",
variables.v_userScreenname);

cmd.Parameters.AddWithValue("?following",
variables.v_friends);

cmd.Parameters.AddWithValue("?follower",
variables.v_followers);
                cmd.ExecuteNonQuery();
                cmd.Parameters.Clear();
            }
            dr.Close();
            con.Close();
```

Kode Sumber 6 *get following* dan *get follower* pengirim

Sedangkan kode sumber untuk mendapatkan teks *tweet* pada isi timeline pengirim ditunjukan pada Kode Sumber 7.

```
IEnumerable<TwitterStatus> tweetuser =
service.ListTweetsOnUserTimeline(new
ListTweetsOnUserTimelineOptions { ScreenName =
akun, Count = 10 });
            try
            {
                foreach (var tweet in tweetuser)
                {
                    //foreach (var tweeturl in
tweet.Entities.Urls)
                    //{
```

```
                            variables.doc2 = new
List<string>();
                            string kalimatnya = "";
                            string urlnya = "";
                            string[] words2 = null;

                            kalimatnya = tweet.Text;
                        }
                    }
```

Kode Sumber 7 mendapatkan *tweet* pada *timeline* pengirim

E. Proses Fitur Pengambilan Nilai Panjang Rantai *Redirect URL*

Proses ini dilakukan pada ekstraksi fitur, dimana terdapat 10 fitur, diantaranya yakni fitur pengambilan nilai panjang rantai *redirect URL*. Pada tahap normalisasi, setiap nilai persamaan antar atribut fitur diproses dengan persamaan yang sudah dijelaskan pada BAB III. Dari hasil normalisasi, akan diproses pengambilan nilai panjang rantai *redirect URL*. Kode Sumber 8 menunjukkan implementasi pengambilan nilai panjang rantai *redirect URL*.

```
foreach (var url in variables.EP_points)
{
      variables.EP_initURL = new List<string>();
    MySqlDataReader dr;
      List<double> posEP = new List<double>();
      List<double> panjang = new List<double>();
            string ambilpos = "SELECT
distinct(urlredirect), posisi, length, urltweet
FROM `urlredirect` WHERE `urlredirect`='" +
entrypoint + "' and
user_training='"+variables.v_usertraining+"'";
            con.Open();
            cmd = con.CreateCommand();
            cmd.CommandText = ambilpos;
            dr = cmd.ExecuteReader();
            while (dr.Read())
            {
                double normalisasiPos = 0;
                double normalisasiLen = 0;
                string ep = dr.GetString(0);
```

```
                double pos = dr.GetDouble(1);
                double length = dr.GetDouble(2);
                string initurl = dr.GetString(3);

                if (length >= 7)
                {
                    length = 7;
                }

                normalisasiLen = length / 7;
                if (normalisasiLen > 0.28 &&
normalisasiLen < 0.29)
                    normalisasiLen = 0.28;
                if (normalisasiLen > 0.42 &&
normalisasiLen < 0.43)
                    normalisasiLen = 0.43;
                if (normalisasiLen > 0.57 &&
normalisasiLen < 0.58)
                    normalisasiLen = 0.57;
                if (normalisasiLen > 0.71 &&
normalisasiLen < 0.72)
                    normalisasiLen = 0.71;
                if (normalisasiLen > 0.85 &&
normalisasiLen < 0.86)
                    normalisasiLen = 0.85;

                 panjang.Add(normalisasiLen);
             }
             dr.close();
             con.close();
       for (int i = 0; i < panjang.Count; i++)
            {
                string updateTraining = "UPDATE " +
variables.v_usertraining + " SET length_url='"+
panjang[i] + "', position_EP='" + posEP[i] + "'
WHERE urltweet='" +  variables.EP_initURL[i] +
"';";

                con.Close();
                con.Open();
                cmd = con.CreateCommand();
                cmd.CommandText = updateTraining;
                cmd.ExecuteNonQuery();
                con.Close();
```

```
            }
}
```

Kode Sumber 8 Fungsi pengambilan nilai panjang *redirect URL*

F. Proses Fitur Pengambilan Nilai Frekuensi *Entrypoint URL*

Proses ini dilakukan pada ekstraksi fitur, dimana terdapat 10 fitur, diantaranya yakni fitur pengambilan nilai frekuensi *entrypoint URL*. Pada tahap normalisasi, setiap nilai persamaan antar atribut fitur diproses dengan persamaan yang sudah dijelaskan pada BAB III. Dari hasil normalisasi, akan diproses pengambilan nilai frekuensi *entrypoint URL*. Kode Sumber 9 menunjukkan implementasi pengambilan nilai frekuensi *entrypoint URL*.

```
for (int i = 0; i < variables.EP_points.Count; i++)
{
                double normalisasiFreq = 0;

                List<string> inisialForFreq = new
List<string>();
                List<string> inisialForDifInit =
new List<string>();
                List<double> normalInisial = new
List<double>();
                MySqlDataReader dr;
                string ambil_inisial = "SELECT
distinct(urltweet) FROM `urlredirect` WHERE
`urlredirect`='" + variables.EP_points[i] + "' and
user_training='"+variables.v_usertraining+"'";
                con.Open();
                cmd = con.CreateCommand();
                cmd.CommandText = ambil_inisial;
                dr = cmd.ExecuteReader();
                while (dr.Read())
                {
                    string initURL =
dr.GetString(0);
                    inisialForFreq.Add(initURL);
                }
                dr.Close();
```

```
                con.Close();
                //for get frequency entrypoint
                normalisasiFreq =
variables.EP_freq[i] / 100;

                foreach (var init in
inisialForFreq)
                {
                    string updateTraining = "UPDATE
" + variables.v_usertraining + " SET
frequencyEP='"+normalisasiFreq+"' WHERE urltweet='"
+ init + "';";
                    con.Close();
                    con.Open();
                    cmd = con.CreateCommand();
                    cmd.CommandText =
updateTraining;
                    cmd.ExecuteNonQuery();
                    con.Close();
                }
}
```

Kode Sumber 9 Fungsi pengambilan nilai frekuensi *entrypoint URL*

G. Proses Fitur Pengambilan Nilai Letak *Entrypoint URL*

Proses ini dilakukan pada ekstraksi fitur, dimana terdapat 10 fitur, diantaranya yakni fitur pengambilan nilai letak *entrypoint URL*. Pada tahap normalisasi, setiap nilai persamaan antar atribut fitur diproses dengan persamaan yang sudah dijelaskan pada BAB III. Dari hasil normalisasi, akan diproses pengambilan nilai letak *entrypoint URL*. Kode Sumber 10 menunjukkan implementasi pengambilan nilai letak dari *entrypoint URL*.

```
foreach (var url in variables.EP_points)
{
    variables.EP_initURL = new List<string>();
    MySqlDataReader dr;
       List<double> posEP = new List<double>();
       List<double> panjang = new List<double>();
            string ambilpos = "SELECT
distinct(urlredirect), posisi, length, urltweet
FROM `urlredirect` WHERE `urlredirect`='" +
entrypoint + "' and
user_training='"+variables.v_usertraining+"'";
```

```
            con.Open();
            cmd = con.CreateCommand();
            cmd.CommandText = ambilpos;
            dr = cmd.ExecuteReader();
            while (dr.Read())
            {
                double normalisasiPos = 0;
                double normalisasiLen = 0;
                string ep = dr.GetString(0);
                double pos = dr.GetDouble(1);
                double length = dr.GetDouble(2);
                string initurl = dr.GetString(3);

                normalisasiPos = pos/length;
                posEP.Add(normalisasiPos);
             }
       for (int i = 0; i < posEP.Count; i++)
       {
          string updateTraining = "UPDATE " +
variables.v_usertraining + " SET length_url='"+
panjang[i] + "', position_EP='" + posEP[i] + "'
WHERE urltweet='" +  variables.EP_initURL[i] +
"';";
                con.Close();
                con.Open();
                cmd = con.CreateCommand();
                cmd.CommandText = updateTraining;
                cmd.ExecuteNonQuery();
                con.Close();
       }
}
```

Kode Sumber 10 pengambilan nilai letak *entrypoint URL*

H. Proses Fitur Pengambilan Nilai Jumlah Inisial *URL*

Proses ini dilakukan pada ekstraksi fitur, dimana terdapat 10 fitur, diantaranya yakni fitur pengambilan nilai jumlah inisial *URL* pada *entrypoint URL* yang sama. Pada tahap normalisasi, setiap nilai persamaan antar atribut fitur diproses dengan persamaan yang sudah dijelaskan pada BAB III. Dari hasil normalisasi, akan diproses pengambilan nilai jumlah inisial *URL* pada *entrypoint*

URL yang sama. Kode Sumber 11 menunjukkan implementasi pengambilan nilai jumlah inisial *URL* pada *entrypoint URL*.

```
for (int i = 0; i < variables.EP_points.Count; i++)
{
  double normalisasiDiffInit = 0;
  string diffInitURL = "SELECT
distinct(urlredirect), urltweet  FROM `urlredirect`
WHERE `urlredirect`='" + variables.EP_points[i] +
"' and user_training='" + variables.v_usertraining
+ "' ";
   con.Open();
   cmd = con.CreateCommand();
   cmd.CommandText = diffInitURL;
   dr = cmd.ExecuteReader();
   while (dr.Read())
   {
        string urlredirect = dr.GetString(0);
        string initURL = dr.GetString(1);
        inisialForDifInit.Add(initURL);
    }
    normalisasiDiffInit = inisialForDifInit.Count /
variables.EP_freq[i];
    dr.Close();
    con.Close();
    for (int j = 0; j < inisialForDifInit.Count;
j++)
    {
         string updateTraining = "UPDATE " +
variables.v_usertraining + " SET
differentInitURL='" + normalisasiDiffInit + "'
WHERE urltweet='" + inisialForDifInit[j] + "';";
          con.Close();
          con.Open();
          cmd = con.CreateCommand();
         cmd.CommandText = updateTraining;
         cmd.ExecuteNonQuery();
         con.Close();
    }
}
```

Kode Sumber 11 mendapatkan nilai jumlah pengirim

I. Proses Fitur Pengambilan Nilai Perbedaan *Landing URL*

Proses ini dilakukan pada ekstraksi fitur, dimana terdapat 10 fitur, diantaranya yakni fitur pengambilan nilai perbedaan *landing URL* pada *entrypoint URL* yang sama. Pada tahap normalisasi, setiap nilai persamaan antar atribut fitur diproses dengan persamaan yang sudah dijelaskan pada BAB III. Dari hasil normalisasi, akan diproses pengambilan nilai perbedaan *landing URL* pada *entrypoint URL* yang sama. Kode Sumber 12 menunjukkan implementasi pengambilan nilai perbedaan *landing URL* pada *entrypoint URL*. Sedangkan Kode Sumber 13 menunjukan implementasi pengecekkan dengan menggunakan peramban normal, sedangkan Kode Sumber 14 menunjukan implementasi pengecekkan dengan menggunakan *crawler*.

```
for (int i = 0; i < variables.EP_points.Count; i++)
{
      variables.urlcek = "";
      variables.browser1 = new List<string>();
      variables.browser2 = new List<string>();
      variables.urlcek = variables.EP_points[i];
      double normalisasiLand = 0;
      variables.browser1.Add(variables.urlcek);
      variables.browser2.Add(variables.urlcek);
      th1 = new Thread(peramban.userAgent1);
      th1.Start();
      th2 = new Thread(peramban.userAgent2);
      th2.Start();
      th1.Join();
      th2.Join();
      double landing = 0;

      try
      {
          for (int j = 0; j
<variables.browser1.Count - 1; j++)
          {
             if (variables.browser1[j] ==
variables.browser2[j])
                 {
                              landing = 1;
```

```
                              continue;
                        }
                        else if
(variables.browser1[j] != variables.browser2[j])
                        {
                              landing = 2;
                              break;
                         }
                   }
             }
             catch
             {
             }

          normalisasiLand = landing /
variables.EP_freq[i];
          MySqlDataReader dr;
          List<string> urlinit = new
List<string>();
          string ambil_initurl = "SELECT
distinct(urltweet) FROM `urlredirect` WHERE
`urlredirect`='" + variables.EP_points[i] + "' and
user_training='" + variables.v_usertraining + "'";
           con.Open();
           cmd = con.CreateCommand();
           cmd.CommandText = ambil_initurl;
           dr = cmd.ExecuteReader();
           while (dr.Read())
           {
                string initURL = dr.GetString(0);
                urlinit.Add(initURL);
            }
            con.Close();

            foreach (var urlxx in urlinit)
            {
               string updateTraining = "UPDATE " +
variables.v_usertraining + " SET
differentLanding='" + normalisasiLand + "' WHERE
urltweet='" + urlxx + "';";
                con.Close();
                con.Open();
                cmd = con.CreateCommand();
                cmd.CommandText = updateTraining;
```

```
                cmd.ExecuteNonQuery();
                con.Close();
            }
}
```

Kode Sumber 12 mendapatkan nilai perbedaan *landing URL* pada *entrypoint URL*

```
public void userAgent1()
{
    try
    {
        StringBuilder sb = new StringBuilder();
        string location =
string.Copy(variables.urlcek);
        while (!string.IsNullOrWhiteSpace(location))
        {
            sb.AppendLine(location);
            HttpWebRequest req =
HttpWebRequest.CreateHttp(location);
            req.UserAgent = "Mozilla/5.0
(compatible; U; ABrowse 0.6; Syllable)
AppleWebKit/420+ (KHTML, like Gecko)";
             req.AllowAutoRedirect = false;
                     //
             using (HttpWebResponse resp =
(HttpWebResponse)req.GetResponse())
             {
                   location =
resp.GetResponseHeader("Location");
variables.browser1.Add(location);
              }
              Thread.Sleep(1000);
           }
        }
        catch
        {
                // MessageBox.Show("{0} tidak dapat
dicek memakai crawler ini", variables.v_urltweet);
        }
}
```

Kode Sumber 13 pengecekkan landing URL dengan menggunakan peramban normal

```
public void userAgent2()
{
            try
            {
                StringBuilder sb = new
StringBuilder();
                string location =
string.Copy(variables.urlcek);
                while
(!string.IsNullOrWhiteSpace(location))
                {
                    sb.AppendLine(location);
                    HttpWebRequest req =
HttpWebRequest.CreateHttp(location);
                    req.UserAgent = "Sogou web
spider/4.0(+http://www.sogou.com/docs/help/webmaste
rs.htm#07)";
                    req.AllowAutoRedirect = false;
                    using (HttpWebResponse resp =
(HttpWebResponse)req.GetResponse())
                    {
                        location =
resp.GetResponseHeader("Location");
                        //listnya yg pertama

variables.browser2.Add(location);
                    }
                    Thread.Sleep(1000);
                }
            }
            catch
            {
            }
}
```

Kode Sumber 14 pengecekkan landing URL dengan menggunakan crawler

J. Proses Fitur Pengambilan Nilai Jumah Pengirim *Entrypoint URL*

Proses ini dilakukan pada ekstraksi fitur, dimana terdapat 10 fitur, diantaranya yakni fitur pengambilan nilai jumlah pengirim *entrypoint URL*. Pada tahap normalisasi, setiap nilai persamaan

antar atribut fitur diproses dengan persamaan yang sudah dijelaskan pada BAB III. Dari hasil normalisasi, akan diproses pengambilan nilai jumlah pengirim *entrypoint URL*. menunjukkan implementasi pengambilan nilai frekuensi *entrypoint URL*. Kode Sumber 15 menunjukkan implementasi pengambilan nilai jumlah pengirim *entrypoint URL*.

```
for (int i = 0; i < variables.EP_points.Count; i++)
{
  MySqlDataReader dr;
                double normalisasi = 0;
                List<string> listIniturl = new
List<string>();
                List<string> listAkun = new
List<string>();

                //get number akun
                string ambil_akun = "SELECT
distinct(`account`) FROM `urlredirect` WHERE
`urlredirect`='" + variables.EP_points[i] + "' and
user_training='" + variables.v_usertraining + "'";
                con.Open();
                cmd = con.CreateCommand();
                cmd.CommandText = ambil_akun;
                dr = cmd.ExecuteReader();
                while (dr.Read())
                {
                     string akunnya =
dr.GetString(0);
                     listAkun.Add(akunnya);
                }
                con.Close();

                // update akun number
                normalisasi = listAkun.Count /
variables.EP_freq[i];
                foreach (var akun in listAkun)
                {
                    string updateLength = "UPDATE "
+ variables.v_usertraining + " SET attack_number='"
+ normalisasi + "' WHERE account='" + akun + "';";
                     con.Close();
```

```
                    con.Open();
                    cmd = con.CreateCommand();
                    cmd.CommandText = updateLength;
                    cmd.ExecuteNonQuery();
                    con.Close();
                }
}
```

Kode Sumber 15 mendapatkan nilai dari jumlah pengirim entrypoint URL

K. Proses Fitur Pengambilan Nilai Simpangan Baku Pada *Follower* Pengirim

Proses ini dilakukan pada ekstraksi fitur, dimana terdapat 10 fitur, diantaranya yakni fitur pengambilan nilai simpangan baku pada *follower* pengirim *entrypoint URL*. Pada tahap normalisasi, setiap nilai persamaan antar atribut fitur diproses dengan persamaan yang sudah dijelaskan pada BAB III. Dari hasil normalisasi, akan diproses pengambilan nilai simpangan baku pada *follower* pengirim *entrypoint URL*. Kode Sumber 16 menunjukkan implementasi pengambilan nilai simpangan baku pada *follower* pengirim *entrypoint URL*.

```
for (int i = 0; i < variables.EP_points.Count; i++)
{
   foreach (var akun in listAkun)
   {
         string ambil_ff = "SELECT follower,
following FROM usertwitter WHERE account ='" + akun
+ "'";
         con.Open();
         cmd = con.CreateCommand();
         cmd.CommandText = ambil_ff;
         dr = cmd.ExecuteReader();
         while (dr.Read())
         {
              followlist = dr.GetDouble(0);
               follow.Add(followlist);
          }
          dr.Close();
          con.Close();
```

```
    }
    double sdfollowers =
follow.StandardDeviation();
    double hasilFollower = sdfollowers / (200 *
Math.Sqrt(follow.Count));
    if (hasilFollower > 1)
    {
          hasilFollower = 1;
    }

    foreach (var akun in listAkun)
    {
       string updateLength = "UPDATE " +
variables.v_usertraining + " SET SD_followers='" +
hasilFollower + "' WHERE account='" + akun + "';";
         con.Close();
         con.Open();
         cmd = con.CreateCommand();
         cmd.CommandText = updateLength;
         cmd.ExecuteNonQuery();
         con.Close();
     }
}
```

Kode Sumber 16 mendapatkan nilai simpangan baku pada follower pengirim

L. Proses Fitur Pengambilan Nilai Simpangan Baku Pada *Friend* Pengirim

Proses ini dilakukan pada ekstraksi fitur, dimana terdapat 10 fitur, diantaranya yakni fitur pengambilan nilai simpangan baku pada *friend* pengirim *entrypoint URL*. Pada tahap normalisasi, setiap nilai persamaan antar atribut fitur diproses dengan persamaan yang sudah dijelaskan pada BAB III. Dari hasil normalisasi, akan diproses pengambilan nilai simpangan baku pada *friend* pengirim *entrypoint URL*. menunjukkan implementasi pengambilan nilai simpangan baku pada *friend* pengirim *entrypoint URL*

```
for (int i = 0; i < variables.EP_points.Count; i++)
{
   foreach (var akun in listAkun)
   {
```

```
          string ambil_ff = "SELECT follower,
following FROM usertwitter WHERE account ='" + akun
+ "'";
          con.Open();
          cmd = con.CreateCommand();
          cmd.CommandText = ambil_ff;
          dr = cmd.ExecuteReader();
          while (dr.Read())
          {
               friendlist = dr.GetDouble(1);
               fren.Add(friendlist);
           }
           dr.Close(); con.Close();
     }
     double sdfriendlist = fren.StandardDeviation();
     double  hasilfriend  =  sdfriendlist  /  (200  *
Math.Sqrt(fren.Count));
     if (hasilfriend > 1)
     {
           hasilfriend = 1;
     }

     foreach (var akun in listAkun)
     {
        string updateLength = "UPDATE " +
variables.v_usertraining + " SET SD_friends='" +
hasilfriend + "' WHERE account='" + akun + "';";
          con.Close();
          con.Open();
          cmd = con.CreateCommand();
          cmd.CommandText = updateLength;
          cmd.ExecuteNonQuery();
          con.Close();
      }
}
```

Kode Sumber 17 mendapatkan nilai simpangan baku pada *friend* pengirim

M. Proses Fitur Pengambilan Nilai Simpangan Baku Pada Rasio Dari *Friend-followers* Pengirim

Proses ini dilakukan pada ekstraksi fitur, dimana terdapat 10 fitur, diantaranya yakni fitur pengambilan nilai simpangan baku

pada rasio dari *friend-follower* pengirim *entrypoint URL*. Pada tahap normalisasi, setiap nilai persamaan antar atribut fitur diproses dengan persamaan yang sudah dijelaskan pada BAB III. Dari hasil normalisasi, akan diproses pengambilan nilai simpangan baku pada rasio dari *friend-follower* pengirim *entrypoint URL*. Kode Sumber 18 menunjukkan implementasi pengambilan nilai simpangan baku pada rasio dari *friend-follower* pengirim *entrypoint URL*

```
for (int i = 0; i < variables.EP_points.Count; i++)
{
   foreach (var akun in listAkun)
   {
         string ambil_ff = "SELECT follower,
following FROM usertwitter WHERE account ='" + akun
+ "'";
         con.Open();
         cmd = con.CreateCommand();
         cmd.CommandText = ambil_ff;
         dr = cmd.ExecuteReader();
         while (dr.Read())
         {
              followlist = dr.GetDouble(0);
              friendlist = dr.GetDouble(1);
                      if (followlist < friendlist)
                        {
                            ratioFFreind =
followlist / friendlist;

ratioFF.Add(ratioFFreind);
                        }
                        else
                        {
                            ratioFFollower =
friendlist / followlist;

ratioFF.Add(ratioFFollower);
                        }

               follow.Add(followlist);
               fren.Add(friendlist);
          }
          dr.Close();
```

```
            con.Close();
        }
        double sdRatio = ratioFF.StandardDeviation();
        double hasilFollower = sdfollowers / (200 *
Math.Sqrt(follow.Count));
        double hasilfriend = sdfriendlist / (200 *
Math.Sqrt(fren.Count));
        if (hasilFollower > 1)
        {
              hasilFollower = 1;
           }
           if (hasilfriend > 1)
            {
                  hasilfriend = 1;
            }
        foreach (var akun in listAkun)
        {
           string updateLength = "UPDATE " +
variables.v_usertraining + " SET SD_followers='" +
hasilFollower + "', SD_friends='" + hasilfriend +
"', SD_ratio_ff='" + hasilrasio + "' WHERE
account='" + akun + "';";
            con.Close();
            con.Open();
            cmd = con.CreateCommand();
            cmd.CommandText = updateLength;
            cmd.ExecuteNonQuery();
            con.Close();
         }
}
```

Kode Sumber 18 mendapatkan nilai simpangan baku rasio pada friend-follower pengirim

N. Proses Fitur Pengambilan Nilai Kemiripan Teks *Tweet* Pada Pengirim

Proses ini dilakukan pada ekstraksi fitur, dimana terdapat 10 fitur, diantaranya yakni fitur pengambilan nilai kemiripan teks *tweet* pada pengirim. Pada tahap normalisasi, setiap nilai persamaan antar atribut fitur diproses dengan persamaan yang sudah dijelaskan pada BAB III. Dari hasil normalisasi, akan diproses pengambilan nilai kemiripan teks *tweet* pada pengirim.

Kode Sumber 19 menunjukkan implementasi pengambilan nilai kemiripan teks *tweet* pada pengirim.

```
variables.doc1 = new List<string>();
double jumlah = 0;
double penyebut = 0;
double Tsimilar = 0;
string[] words = null;

words =
variables.v_tweettext.Split(variables.delimikaliamt
);

//MessageBox.Show(variables.v_tweettext);

            foreach (string s in words)
            {
                //Console.WriteLine(s);
                variables.doc1.Add(s);

}twit.getTweetUser(variables.v_userScreenname);
            jumlah = variables.pembilang.Sum();
            penyebut =
Math.Sqrt(variables.jumlahkuadrat.Sum());
            Tsimilar = jumlah / penyebut;
            //MessageBox.Show(Tsimilar.ToString());
            if (Tsimilar < 0.1)
            {
                Tsimilar = 0;
            }
            if (Tsimilar < 0.2 && Tsimilar >=0.1)
            {
                Tsimilar = 0.1;
            }
            if (Tsimilar < 0.3 && Tsimilar >= 0.2)
            {
                Tsimilar = 0.2;
            }
            if (Tsimilar < 0.4 && Tsimilar >= 0.3)
            {
                Tsimilar = 0.3;
            }
            if (Tsimilar < 0.5 && Tsimilar >= 0.4)
            {
```

```
                Tsimilar = 0.4;
            }
            if (Tsimilar < 1 && Tsimilar >= 0.5)
            {
                Tsimilar = 0.5;
            }
            if (Tsimilar > 1)
            {
                Tsimilar = 1;
            }
           string updateTraining = "UPDATE " +
variables.v_usertraining + " SET text_similarity =
'" + Tsimilar + "' WHERE urltweet='" +
variables.v_urltweet + "';";
            con.Open();
            cmd = con.CreateCommand();
            cmd.CommandText = updateTraining;
            cmd.ExecuteNonQuery();
            con.Close();
```

Kode Sumber 19 mendapatkan nilai kemiripan teks tweet pada pengirim

O. Proses Klasifikasi Fitur Dengan Menggunakan Algoritma *Decision Tree*

Proses ini dilakukan klasifikasi fitur dengan menggunakan algoritma *decision tree* berdasarkan model *tree* yang telah dibangun. Kode Sumber 20 menunjukkan implementasi klasifikasi fitur dengan menggunakan algoritma *decision tree*.

```
List<string> isiurls = new List<string>();
List<string> labeling = new List<string>();

MySqlDataReader dr;
            string labelin = "SELECT
`urltweet`,`differentLanding`,`position_EP`,`differ
entInitURL`,`text_similarity`,`SD_followers` FROM "
+ variables.v_usertraining + " WHERE flag = 1 and
label=''";
            con.Open();
            cmd = con.CreateCommand();
            cmd.CommandText = labelin;
            dr = cmd.ExecuteReader();
            while (dr.Read())
```

```
            {
                string label = "";
                string urls = dr.GetString(0);
                double difLanding =
dr.GetDouble(1);
                double position_EP =
dr.GetDouble(2);
                double difInitURL =
dr.GetDouble(3);
                double Tsimilars = dr.GetDouble(4);
                double SD_followers =
dr.GetDouble(5);

                if( difLanding > 0.090909)
                {
                    if (position_EP >
0.666666666666667)
                    {
                        if (Tsimilars > 0)
                        {
                            if (SD_followers >
0.473762)
                                label = "AMAN";
                            else
                                label =
"BERBAHAYA";
                        }
                        else if(Tsimilars <= 0)
                            label = "AMAN";
                    }
                    else if (position_EP <=
0.666666666666667)
                    {
                        if (difInitURL > 0.25)
                            label = "AMAN";
                        else if (difInitURL <=
0.25)
                        {
                            if (difLanding > 0.4)
                                label =
"BERBAHAYA";
                            else if (difLanding <=
0.4)
                                label = "AMAN";
```

```
                            }
                        }
                    }
                    else if(difLanding <= 0.090909)
                        label = "BERBAHAYA";
                    isiurls.Add(urls);
                    labeling.Add(label);
        }
for (int i = 0; i < isiurls.Count; i++)
{
                    string updateLength = "UPDATE " +
variables.v_usertraining + " SET label='" +
labeling[i] + "' WHERE urltweet='" + isiurls[i] +
"';";
                    con.Close();
                    con.Open();
                    cmd = con.CreateCommand();
                    cmd.CommandText = updateLength;
                    cmd.ExecuteNonQuery();
                    con.Close();
}
```

Kode Sumber 20 implementasi algoritma decision tree

P. Proses Peringatan Pada Peramban Mozilla Firefox Menggunakan Greasemonkey *add-ons*

Proses ini dilakukan peringatan pada peramban Mozilla Firefox dengan menggunakan Greasemonkey *add-ons*. Hal ini bertujuan untuk memberikan peringatan apabila terdapat *URL* yang berbahaya pada *timeline* pengguna maupun *mention* pengguna. Kode Sumber 21 menunjukkan implementasi proses pengambilan label yang merupakan *URL* berbahaya. Sedangkan Kode Sumber 22 menunjukan implementasi format penulisan *javascript* peringatan pada peramban Mozilla Firefox menggunakan Greasemonkey *add-ons*.

```
MySqlDataReader dr;
            userShows = new List<string>();
            urlShows = new List<string>();
            tweetShows = new List<string>();
            string ambil_penyerang = "SELECT
u.account, u.urltweet, u.tweet FROM
"+variables.v_username+" u,
```

```
"+variables.v_usertraining+" t WHERE
t.label='BERBAHAYA' and u.urltweet = t.urltweet";
            con.Open();
            cmd = con.CreateCommand();
            cmd.CommandText = ambil_penyerang;
            dr = cmd.ExecuteReader();
            if (dr.HasRows == true)
            {
                Console.WriteLine("ada penyerang
mengirimkan URL berbahaya!");
                while (dr.Read())
                {
                    userShow = dr.GetString(0);
                    urlShow = dr.GetString(1);
                    tweetShow = dr.GetString(2);

                    userShows.Add(userShow);
                    urlShows.Add(urlShow);
                    tweetShows.Add(tweetShow);
                }
                format(namacom);
            }
            else
            {
                Console.WriteLine("tidak ada
penyerang yang mengirimkan URL berbahaya");
            }
            con.Close();
```

Kode Sumber 21 pengambilan label URL berbahaya

```
string[] path =
Directory.GetDirectories("C:\\Users\\"+namacom+"\\A
ppData\\Roaming\\Mozilla\\Firefox\\Profiles\\",
"*.default");
            using (StreamWriter save = new
StreamWriter(path[0]+"\\gm_scripts\\detectURL\\dete
ctURL.user.js"))
            {
                save.WriteLine("//
==UserScript==");
                save.WriteLine("// @name
detectURL");
```

```
                save.WriteLine("// @namespace
idham.greasemonkey.APENURL");
                save.WriteLine("// @description
alert when url has been detect on tweet!");
                save.WriteLine("// @include
https://twitter.com");
                save.WriteLine("// @include
http://twitter.com");
                save.WriteLine("// @include
https://twitter.com/*");
                save.WriteLine("// @include
http://twitter.com/*");
                save.WriteLine("// @include
https://www.twitter.com/*");
                save.WriteLine("// @include
http://www.twitter.com/*");
                save.WriteLine("// @version
1");
                save.WriteLine("// @grant
none");
                save.WriteLine("//
==/UserScript==");
                save.WriteLine("var els =
document.getElementsByTagName('a');");
                save.WriteLine("for (var i = 0, l =
els.length; i < l; i++) {");
                save.WriteLine("    var el =
els[i];");
                for (int i = 0; i < urlShows.Count;
i++)
                {
                    save.WriteLine("if(el.href ===
'"+urlShows[i]+"') {");
                    save.WriteLine("alert('UPS!
tweetnya " + userShows[i] + " seperti >>" +
tweetShows[i] + "<< is MALICIOUS!');");
                    save.WriteLine("}");
                }
                save.WriteLine("}");
            }
```

Kode Sumber 22 format javascript untuk menindih skrip Greasemonkey add-ons

Q. Proses Pengiriman Notifikasi Melalui *E*-mail

Proses ini dilakukan pengiriman notifikasi mlalui *e-mail*. Hal ini bertujuan untuk memberikan notifikasi apabila pengguna sedang meninggalkan program, atau sebagai histori untuk mencatat *URL* apa sajakah yang berbahaya. Kode Sumber 23 menunjukkan pengambilan label yang merupakan *URL* berbahaya, sedangkan Kode Sumber 24 menunjukan implementasi format *e-mail* untuk pengiriman notifikasi pada pengguna.

```
MySqlDataReader dr;
            string ambil_penyerang = "SELECT
u.account, u.crawl_type, u.urltweet, u.tweet FROM
"+variables.v_username+" u,
"+variables.v_usertraining+" t WHERE
t.label='BERBAHAYA' and u.urltweet = t.urltweet";
            con.Open();
            cmd = con.CreateCommand();
            cmd.CommandText = ambil_penyerang;
            dr = cmd.ExecuteReader();
            if (dr.HasRows == true)
            {
                Console.WriteLine("terdapat
penyerang, silahkan cek notifikasi email anda!");
                while (dr.Read())
                {
                    userattack = dr.GetString(0);
                    typeattack = dr.GetString(1);
                    urlattack = dr.GetString(2);
                    tweetattack = dr.GetString(3);
                    userattacks.Add(userattack);
                    typeattacks.Add(typeattack);
                    urlattacks.Add(urlattack);
                    tweetattacks.Add(tweetattack);
                }
                email(useremail);
            }
            else
            {
                Console.WriteLine("Tidak Ada
Penyerang!");
            }
            con.Close();
```

Kode Sumber 23 pengambilan label *URL* berbahaya

```
public void email(string useremail)
{
            //MessageBox.Show("ada penyerang di
Timeline/Mention anda! silahkan cek di notifikasi
Email anda");
            using (StreamWriter save = new
StreamWriter("FormatEmail.txt"))
            {
                string pengantar = "Hallo
"+variables.v_username+"! berikut daftar Attacker
yang ada pada timeline/mention anda dimana telah
menyebarkan tweet yang disertai url yang
dikategorikan berbahaya:";
                save.WriteLine(pengantar);
                for (int i = 0; i <
userattacks.Count; i++)
                {
                    save.WriteLine("Akun:
"+userattacks[i]);
                    save.WriteLine("Detect On: " +
typeattacks[i]);
                    save.WriteLine("url:
"+urlattacks[i]);
                    save.WriteLine("tweet:
"+tweetattacks[i]);

save.WriteLine("==================================
==================");
                }
            }

            string subject = "[NO REPLY] Your
Timeline/Mention Has Some Malicious Tweets!";
            string body =
File.ReadAllText("FormatEmail.txt");

            MailMessage msg = new MailMessage();

            msg.From = new
MailAddress("apenurl.apps@gmail.com");
            msg.To.Add(useremail);
            msg.Subject = subject;
            msg.Body = body;
            //msg.Priority = MailPriority.High;
```

```
            using (SmtpClient client = new
SmtpClient())
            {
                client.EnableSsl = true;
                client.UseDefaultCredentials =
false;
                client.Credentials = new
NetworkCredential("apenurl.apps@gmail.com",
"tukangngompol62");
                client.Host = "smtp.gmail.com";
                client.Port = 587;
                client.DeliveryMethod =
SmtpDeliveryMethod.Network;

                client.Send(msg);
            }
}
```

Kode Sumber 24 format pengiriman notifikasi melalui *e-mail*

R. Daftar Pemodelan *Tree*

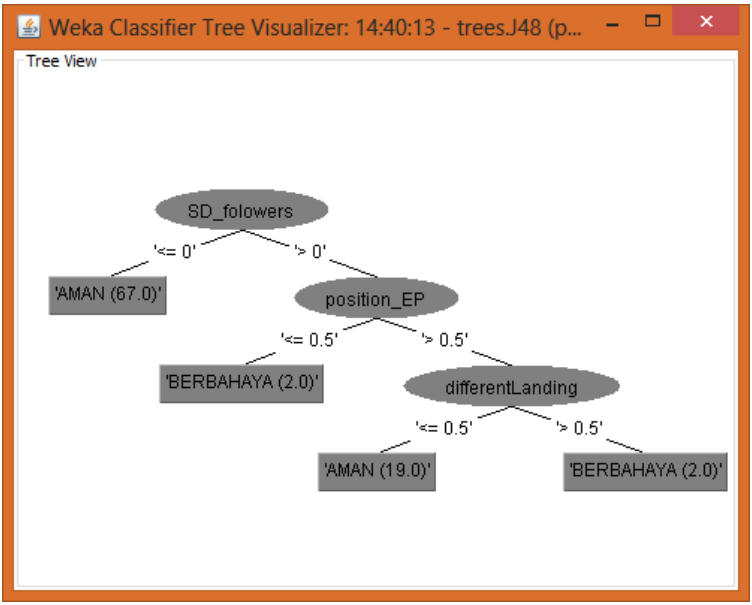
Berikut merupakan pemodelan *tree* yang dibentuk berdasarkan aplikasi WEKA. Terdapat lima model *tree* yang dibangun didasari pada jumlah data *training tweet* yang telah dikumpulkan. Berikut merupakan daftar pemodelan *tree* tersebut.

1. Pemodelan A (100 Data *Training*)

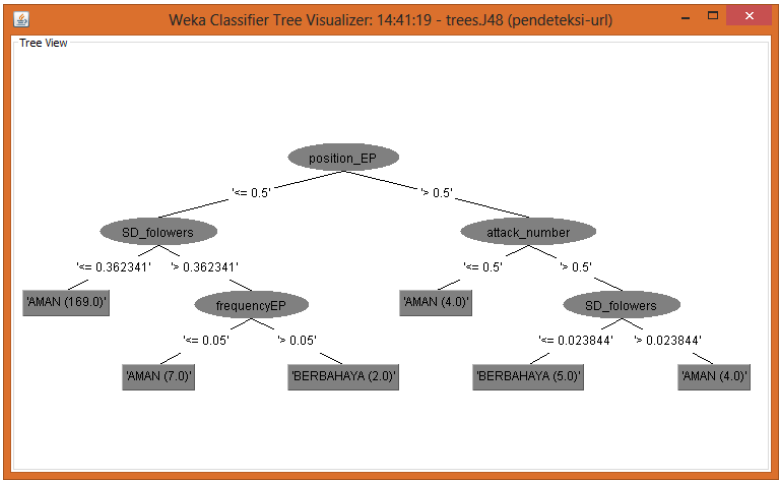
Pemodelan A mempunyai jumlah data sebanyak 100 data, pada pengujian *10 cross validation*, mempunyai akurasi sebesar 95.55%. data tersebut hanya mengambil beberapa fitur diantaranya SD_followers, position_EP, differentLanding. Hal ini ditunjukan pada Gambar 0.1.

2. Pemodelan B (200 Data *Training*)

Pemodelan B mempunyai jumlah data sebanyak 200 data, pada pengujian *10 cross validation*, mempunyai akurasi sebesar 96.85%. data tersebut hanya mengambil beberapa fitur diantaranya position_EP, SD_followers, attack_number, frequencyEP. Hal ini ditunjukan pada Gambar 0.2.



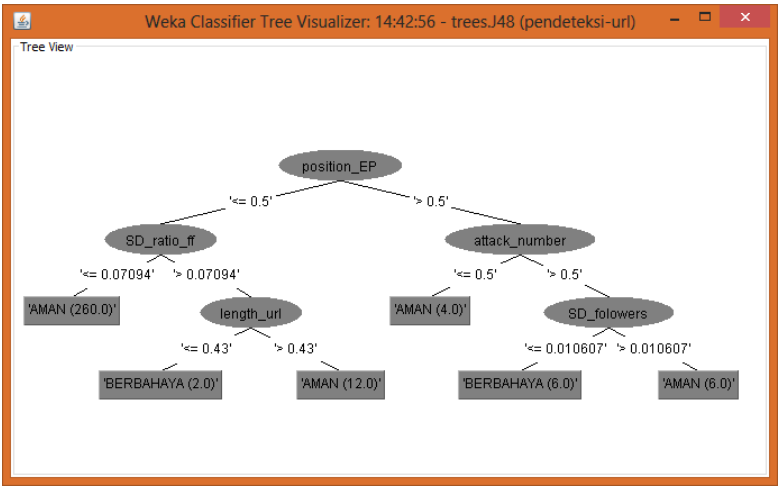
Gambar 0.1 Pemodelan *Tree* pada pemodelan A



Gambar 0.2 Pemodelan *Tree* pada pemodelan B

3. Pemodelan C (300 Data *Training*)

Pemodelan C mempunyai jumlah data sebanyak 300 data, pada pengujian *10 cross validation*, mempunyai akurasi sebesar 97.93%. data tersebut hanya mengambil beberapa fitur diantaranya position_EP, SD_followers, attack_number, length_url, SD_ratio_ff. Hal ini ditunjukan pada Gambar 0.3.



Gambar 0.3 Pemodelan *Tree* pada pemodelan C

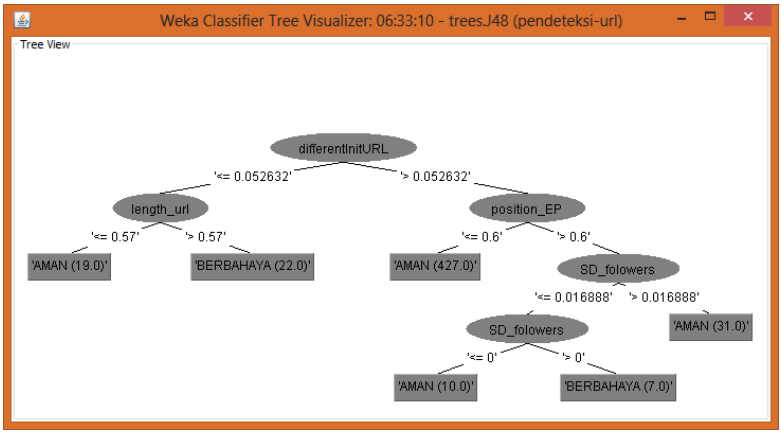
4. Pemodelan D (500 Data *Training*)

Pemodelan D mempunyai jumlah data sebanyak 500 data, pada pengujian *10 cross validation*, mempunyai akurasi sebesar 99.598%. data tersebut hanya mengambil beberapa fitur diantaranya differentInitURL, position_EP, SD_followers. Hal ini ditunjukan pada Gambar 0.4.

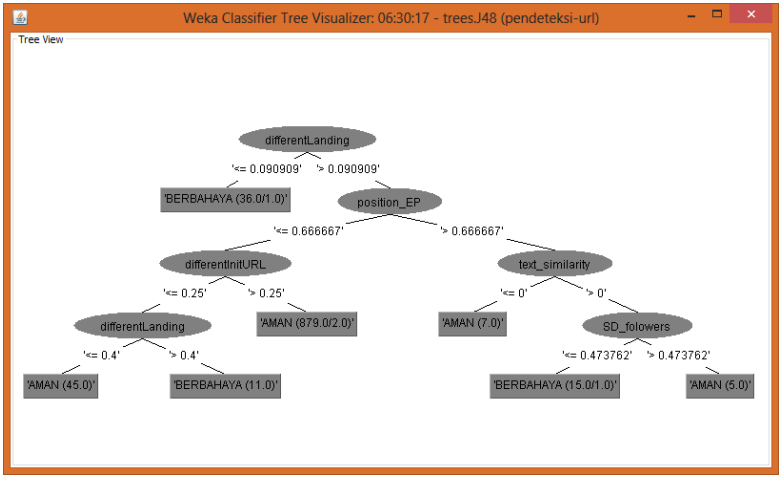
5. Pemodelan E (1000 Data *Training*)

Pemodelan E mempunyai jumlah data sebanyak 1000 data, pada pengujian *10 cross validation*, mempunyai akurasi sebesar 99.599%. data tersebut hanya mengambil beberapa fitur diantaranya differentLanding, position_EP, differentInitURL,

text_similarity, SD_followers. Hal ini ditunjukan pada Gambar 0.5.



Gambar 0.4 Pemodelan *Tree* pada pemodelan D



Gambar 0.5 Pemodelan *Tree* pada pemodelan E

BIODATA PENULIS

Idham Mardi Putra, biasa dipanggil Idham, dilahirkan di Surabaya pada tanggal 6 Juli 1992. Penulis adalah anak ragil dari dua bersaudara. Penulis telah menempuh pendidikan formal di SD Jajar Tunggal III Surabaya (1998-2004), SMP Negeri 28 Surabaya (2004-2007), dan SMA Negeri 15 Surabaya (2007-2010). Pada tahun 2010 penulis diterima di strata satu Jurusan Teknik Informatika Fakultas Teknologi Informasi, Institut Teknologi Sepuluh Nopember Surabaya angkatan 2010 yang terdaftar dengan NRP. 5110100024. Di Jurusan Teknik Informatika ini, penulis mengambil bidang minat *Net Centric Computing* (NCC). Selama menempuh kuliah, penulis aktif sebagai staf departemen pengembangan sumber daya mahasiswa pada tahun kedua, dan pada tahun ketiga sebagai *Instructor Committee* kaderisasi departemen pengembangan sumber daya mahasiswa di Himpunan Mahasiswa Teknik Computer (HMTC). Penulis juga terlibat pada kegiatan Schematics 2012 sebagai Koordinator Keamanan dan Perjinian. Penulis juga aktif sebagai administrator laboratorium Arsitektur dan Jaringan Komputer selama masa perkuliahannya. Pada beberapa acara kampus, penulis juga aktif menjadi panitia, baik sebagai anggota maupun koordinator. Selama masa perkuliahan penulis pernah berkesempatan menjadi asisten praktikum pada mata kuliah Sistem Operasi, dan Jaringan Komputer. Penulis dapat dihubungi melalui alamat *e-mail* di idhammardy024@gmail.com.